Universidade de Brasília
Faculdade de Tecnologia

Aplicabilidade de Técnicas de Inteligência Artificial na Análise Automática de Imagens Agrícolas Aéreas

João Pedro Rebeschini
Gabriel Fernandes Carvalho

PROJETO FINAL DE CURSO
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasília
2023

Universidade de Brasília
Faculdade de Tecnologia

Aplicabilidade de Técnicas de Inteligência Artificial na Análise Automática de Imagens Agrícolas Aéreas

João Pedro Rebeschini
Gabriel Fernandes Carvalho

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Orientador: Prof. Dr. Jones Yudi Mori Alves da Silva

Brasília
2023

Rebeschini, João Pedro.
R291a Aplicabilidade de Técnicas de Inteligência Artificial na Análise Automática de Imagens Agrícolas Aéreas / João Pedro Rebeschini; Gabriel Fernandes Carvalho; orientador Jones Yudi Mori Alves da Silva. -- Brasília, 2023.
100 p.

Projeto Final de Curso (Engenharia de Controle e Automação) -- Universidade de Brasília, 2023.

1. Deep Learning. 2. Visão Computacional. 3. Redes Neurais Convolucionais. 4. Agricultura de Precisão. I. Fernandes Carvalho, Gabriel. II. Rebeschini, João Pedro. III. Mori Alves da Silva, Jones Yudi, orient. IIII. Aplicabilidade de Técnicas de Inteligência Artificial na Análise Automática de Imagens Agrícolas Aéreas

Universidade de Brasília
Faculdade de Tecnologia

Aplicabilidade de Técnicas de Inteligência Artificial na Análise Automática de Imagens Agrícolas Aéreas

João Pedro Rebeschini
Gabriel Fernandes Carvalho

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Trabalho aprovado. Brasília, 21 de julho de 2023:

Prof. Dr. Jones Yudi Mori Alves da Silva,
UnB/FT/ENM
Orientador

Prof. Dr. Gerardo Antonio Idrobo Pizo,
UnB/FGA
Examinador interno

Prof. Dr. André Carmona Hernandes,
UFSCar/DEE
Examinador externo

Brasília
2023

Aos persistentes promotores do desenvolvimento humano.

João Pedro Rebeschini

Dedico este trabalho a todos que, com amor e apoio, tornaram possível esta conquista: minha família, meus amigos e minha namorada Jenyffer.

Gabriel Fernandes Carvalho

Agradecimentos

Aos meus familiares que, em primeira instância, ressalto meus pais, Daniel e Cristiane. Obrigado pelo apoio irrestrito que me foi dado em todas as etapas de minha vida. Somente sob vossa égide, fui capaz de me desenvolver como humano e realizar essa conquista. Aos meus irmãos, Daniel e Marco Antonio, pelas risadas compartilhadas e pelos momentos genuínos de alegria. À minha namorada, Maria, sem cujo apoio tudo teria sido mais difícil nessa etapa final. Seu constante encorajamento e a fé que tem por mim se provaram fortificantes nessa jornada. Aos meus valorosos amigos que fiz ao longo da vida e na universidade que, mesmo sem citações nominais, tornaram esse desafio mais leve e carregado de felicitações. Nesse ínterim, ressalvo o nome de Gabriel, grande companheiro nesse trabalho e em quase toda minha graduação. Também gostaria de agradecer à todo o corpo docente e funcionários da educação, cujo impacto na sociedade é e sempre será inestimável. Em especial, agradeço também ao professor e orientador desse trabalho, Jones Yudi, que tornou a realização desse trabalho possível. Espero que esse pequeno parágrafo possa servir como um vislumbre dos meus mais sinceros agradecimentos àqueles que cito nessa seção.

João Pedro Rebeschini

Primeiramente, gostaria de agradecer aos meus pais, Heider e Luanda, pela constante presença e apoio durante toda minha trajetória acadêmica. Também agradeço aos meus irmãos, Gustavo e Lucas, por sempre estarem disponíveis para me ajudar e compartilhar momentos de alegria. Gostaria também de agradecer aos meus amigos, tanto aqueles que conheci fora da universidade quanto os que fiz durante a graduação, os quais foram fundamentais para que eu pudesse chegar até aqui. E ao João Pedro, minha dupla neste trabalho, agradeço por ser um grande amigo dentro e fora da universidade. Gostaria de agradecer aos professores da UnB por contribuírem significativamente para o meu desenvolvimento profissional e pessoal. Em especial, ao professor Jones Yudi, agradeço por sua orientação e ajuda na realização deste trabalho. De modo especial, agradeço à minha namorada, Jenyffer. Seu apoio, paciência e amor foram fundamentais para que eu pudesse concluir este curso. Nos momentos difíceis sempre me deu forças para continuar e nos bons momentos sempre estava lá para comemorar comigo. Para todos que mencionei, e mesmo aqueles que não, mas que de alguma forma contribuíram para esta conquista, deixo aqui o meu muito obrigado.

Gabriel Fernandes Carvalho

"Anything that could give rise to smarter-than-human intelligence—in the form of Artificial Intelligence, brain-computer interfaces, or neuroscience-based human intelligence enhancement – wins hands down beyond contest as doing the most to change the world. Nothing else is even in the same league."
(Eliezer Yudkowsky)

Resumo

Técnicas de Inteligência Artificial têm alavancado o desempenho do setor agrícola, principalmente devido à introdução de tecnologias como mapeamento via drone, detecção de pestes e no auxílio às tomadas de decisão. Nesse sentido, o presente trabalho tem como objetivo aplicar técnicas de *Deep Learning* em um *dataset* de lavouras de sorgo, tendo como meta detectar as linhas de plantio, calcular o percentual do solo cultivado e identificar falhas na plantação. Técnicas de *Data Augmentation* e *Synthetic Data* foram utilizadas de modo a aumentar e diversificar o *dataset* na tentativa de gerar um modelo final mais robusto e generalista. Outrossim, foi realizado um comparativo entre três arquiteturas de detecção de objetos de alto desempenho (*Faster R-CNN*, *RetinaNet* e YOLO) e suas respectivas variações. Além disso, para a arquitetura vencedora YOLOv8, foram testadas várias profundidades com o objetivo de verificar uma possível correlação entre a precisão média e o tamanho da rede, a qual foi constatada pelo melhor desempenho da YOLOv8 *Extra Large*. Na sequência, testamos também variações de hiperparâmetros como *batch size*, *learning rate* e o otimizador, de onde foi possível obter o melhor conjunto: *batch size* de 16, *learning rate* de 0.001 e SGD como otimizador, aumentando a precisão média da rede para 89.2%, superando em 4% o modelo não otimizado. Algoritmos heurísticos foram desenvolvidos em Python de modo a identificar as falhas e calcular o aproveitamento do solo. Por fim, constatou-se que a inclusão dos dados sintéticos no conjunto de dados de treinamento piorou a performance do modelo otimizado em 4% no que diz respeito à precisão média.

Palavras-chave: Deep Learning. Visão Computacional. Redes Neurais Convolucionais. Agricultura de Precisão.

Abstract

Artificial Intelligence techniques have boosted the performance of the agricultural sector, primarily due to the introduction of technologies such as drone mapping, pest detection, and decision-making support. In this regard, the present work aims to apply Deep Learning techniques to a sorghum crop dataset, with the goal of detecting planting lines, calculating the percentage of cultivated soil, and identifying faults in the plantation. Data Augmentation and Synthetic Data techniques were used to increase and diversify the dataset in an attempt to generate a more robust and generalized final model. In addition, a comparison was made between three high-performance object detection architectures (Faster R-CNN, RetinaNet, and YOLO) and their respective variations. Furthermore, for the winning architecture, YOLOv8, various depths were tested to investigate a possible correlation between average precision and network size, which was confirmed by the superior performance of YOLOv8 Extra Large. Subsequently, we also tested variations of hyperparameters such as batch size, learning rate, and the optimizer, from which the best set was obtained: batch size of 16, learning rate of 0.001, and SGD as the optimizer, increasing the network's average precision to 89.2%, surpassing the unoptimized model by 4%. Heuristic algorithms were developed in Python to identify faults and calculate soil utilization. Finally, it was found that the inclusion of synthetic data in the training dataset degraded the performance of the optimized model by 4% in terms of average precision.

Keywords: Deep Learning. Computer Vision. Convolutional Neural Networks. Precision Agriculture.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AI	Artificial Intelligence	16
ANN	Artficial Neural Network	20
AP	Average Precision	52
BCE	Binary Cross Entropy	39
CE	Cross Entropy	30
CIoU	Complete-IoU	39
CNN	Convolutional Neural Network	24
CSP	Cross-Stage-Partial-connections	34
DFL	Distribution Focal Loss	36
DIP	Digital Image Processing	18
E-ELAN	Extended Efficient Layer Aggregation Network	35
FAIR	Facebook AI Research	28
FN	False Negative	52
FP	False Positive	52
FPN	Feature Pyramid Network	32
GAN	Generative Adversarial Network	40
GIS	Geographic Information System	17
GNSS	Global Navigation Satellite System	16
HOG	Histograma de Gradientes Orientados	28
IoU	Intersect Over Union	52
mAP	Mean Average Precision	52
ML	Machine Learning	16
NLP	Natural Language Processing	20
PANet	Path Aggregation Network	33
R-CNN	Region-based Convoluitonal Neural Network	28
RPAS	Remotely Piloted Aircraft System	18
RPN	Regional Proposal Network	29
SAMA	South Atlantic Magnetic Anomaly	16
SPP	Spatial Pyramid Pooling	33
SVM	Support Vector Machine	28
TN	True Negative	52
TP	True Positive	52
UAV	Unmanned Aerial Vehicle	16
VFL	Varifocal Loss	36
VGG	Visual Geometry Group	32
YOLO	You Only Look Once	32

Lista de símbolos

Símbolos romanos

Símbolos gregos

Sumário

1 Introdução

A introdução de técnicas de Inteligência Artificial (AI, na sigla em ingês) e Veículos Aéreos Não Tripulados (UAVs, na sigla em inglês) tem fomentado uma agricultura moderna cada vez mais eficiente, fazendo com que esse setor experiencie uma grande revolução tecnológica. Por sua vez, tais inovações estão influenciando, de maneira direta, no manejo, gerenciamento e monitoramento das lavouras por parte dos agricultores, o que aumenta o desempenho das mesmas e impulsiona a produção.

Uma das ramificações da AI, o *machine learning* (ML), é uma grande ferramenta na ánalise de imagens áreas agrícolas. Ele tem sido empregado em uma variedade de aplicações, incluindo a identificação e classificação de vegetação, mapeamento das plantações e detecções de ervas daninhas, doenças e deficiências nutricionais nas culturas. (MAKTAB DAR OGHAZ et al., 2019)

Além disso, o uso de máquinas automatizadas para colheita e cultivo tem se tornado cada vez mais popular. No entanto, essas máquinas podem causar danos significativos aos cultivos, especialmente se não são utilizadas ou calibradas adequadamente, podendo compactar o solo e afetar o crescimento e desenvolvimento das plantas . O esmagamento de culturas por máquinas automatizadas pode resultar em perdas financeiras significativas para os agricultores, bem como afetar a qualidade e a quantidade dos produtos colhidos (DAIGH; DEJONG-HUGHES; ACHARYA, 2020). Ademais, pode também causar problemas ambientais, como a erosão do solo e a degradação da qualidade da água. É importante que os agricultores e os fabricantes de máquinas trabalhem juntos para desenvolver tecnologias e práticas que minimizem esses danos (GÜRSOY, 2021).

A principal causa do esmagamento da plantação é a falha de precisão nos veículos autônomos guiados pelo GNSS (Global Navigation Satellite System) (KRÜGER; SPRINGER; LECHNER, 1994), as quais se agravam ainda mais devido a fenômenos naturais como a Anomalia Magnética do Atlântico Sul (SAMA, na sigla em inglês) (ABDU et al., 2005), na qual o Brasil é bastante afetado.

Neste trabalho, exploraremos ainda mais a aplicabilidade das técnicas de AI em imagens aéreas agrícolas, com foco em detectar linhas de plantio, calcular percentual de área cultivada e identificar falhas nas linhas de plantação. Através de análises e discussões, buscamos contribuir para o crescente corpo de pesquisa neste campo emergente e vital.

1.1 Evolução das Técnicas Agrícolas

A demanda por alimentos tem aumentado devido ao crescimento da população mundial e ao aumento do poder aquisitivo em países emergentes. A previsão é que essa tendência continue no futuro, o que colocará uma pressão significativa sobre os sistemas alimentares globais para aumentar a produção agrícola e a eficiência do uso dos recursos (TILMAN et al., 2011).

1.1.1 Contexto Histórico

A busca por novas e melhores técnicas de cultivo faz parte da história da humanidade, tendo sido fundamental para a criação das modernas civilizações. Desde os primeiros povos sedentários, a humanidade tem buscado inovações na agricultura, passando pela criação do arado à utilização da tração animal. O aumento do volume de produção resultante dessas inovações levou ao crescimento populacional e a novas demandas por alimentos, gerando um grande ciclo de ocupação de novas áreas pela agricultura (BOSERUP, 1965). Com o avanço da ciência, o impacto das ações humanas sobre o ambiente tornou-se mais conhecido e entendido, surgindo a necessidade de um aumento de produtividade, ou seja: produzir mais e melhor sem aumentar as áreas de cultivo.

1.1.2 Mecanização do Campo

As revoluções industriais trouxeram a modernização não só nas indústrias e fábricas das cidades, mas também ao campo, por meio de máquinas, implementos e insumos cada vez melhores. A mecanização do campo, ocorrida fortemente na segunda metade do século XX, trouxe melhorias significativas na produtividade. Por meio da mecanização, surgiram tecnologias avançadas como por exemplo a agricultura de precisão e os Sistemas de Informação Geográfica (GIS, na sigla em inglês) (MANI et al., 2021).

1.1.3 Automação no Campo e Agricultura de Precisão

A automação no campo tem dado origem à agricultura de precisão, uma abordagem que utiliza tecnologias avançadas, como sensores, drones, robótica e GIS para aumentar a eficiência e a produtividade da agricultura. Estas tecnologias permitem aos agricultores coletar e analisar dados precisos sobre o solo, as plantas e o clima, e usá-los para tomar decisões informadas (ANDREO, 2013).

A agricultura de precisão oferece muitos benefícios, incluindo aumento da produtividade, redução de custos, melhoria da qualidade dos produtos e proteção ambiental. No entanto, há alguns desafios e problemas associados a ela. Um deles é a precisão da posição, pois a agricultura de precisão depende muito dos GNSS, como GPS, mas esses sistemas podem sofrer interferências, como a SAMA, que pode afetar a precisão da posição. Além disso,

a falta de infraestrutura de banda larga e acesso limitado à tecnologia são outros desafios que podem impedir a adoção da agricultura de precisão em algumas regiões (ONYEMA et al., 2021).

1.1.4 Imageamento Aéreo

O imageamento aéreo, geralmente realizado a partir de uma altitude elevada, é usado para coletar imagens de áreas extensas de terra. Esta técnica é amplamente utilizada na agricultura. As imagens aéreas podem ser feitas usando diferentes meios, como satélites, aviões, ou Aeronaves Pilotadas Remotamente (RPAS, na sigla em inglês). Cada meio tem suas próprias vantagens e desvantagens em termos de resolução, precisão, cobertura e custo (MATESE et al., 2015).

O potencial de utilização do imageamento aéreo na agricultura é amplo, pois ele permite aos agricultores coletar dados precisos sobre o solo, as plantas e o clima. Esses dados podem ser usados para decidir sobre plantio e colheita por exemplo. Além disso, o imageamento aéreo também pode ser usado para monitorar o progresso do cultivo (TSOUROS et al., 2019), identificar problemas como pragas e doenças, e avaliar a eficiência de práticas agrícolas específicas.

No entanto, há alguns problemas e desafios inerentes ao uso do imageamento aéreo na agricultura. Um deles é a precisão dos dados, pois a qualidade das imagens pode ser afetada por fatores como a resolução, a iluminação e a posição do sol. Além disso, a interpretação das imagens pode ser complicada e requerer conhecimento especializado. Outros desafios são o alto custo para a aquisição, a manutenção dos equipamentos e a infraestrutura necessária para processar os dados (LIU et al., 2022).

1.2 Análise Automática de Imagens

O Processamento Digital de Imagens (DIP, na sigla em inglês) surgiu com o objetivo de melhorar as informações presentes nas imagens para interpretação humana, além de processar os dados das imagens para armazenamento, transmissão e representação, permitindo que modelos autônomos os interpretem. Segundo Gonzalez e Woods (2018), o DIP é definido como a utilização de técnicas computacionais para o tratamento e aprimoramento de informações contidas nos dados de uma imagem. Embora não exista uma distinção clara entre o DIP e áreas correlatas como visão computacional e análise de imagens, é possível delimitar que o processamento de imagens se refere a sistemas em que tanto as entradas quanto as saídas são dados pictóricos. Quando um sistema tem como saída dados não-pictóricos, como números ou textos, o mesmo potencialmente se refere a outro campo de estudo.

Não obstante, é importante definir a visão computacional como um campo de estudo da AI que busca equipar computadores com habilidades sensoriais similares às humanas,

permitindo-lhes perceber o ambiente, coletar e compreender dados, tomar decisões e aprender para aprimorar seu desempenho futuro (SEBE et al., 2005). A internet, ao facilitar a transmissão e circulação de vários tipos de dados - incluindo imagens e vídeos - tem catalisado o interesse na visão computacional. Isso se deve à possibilidade de acesso a um volume maior de dados para treinamento e à aplicação de modelos inteligentes que aceleram os processos. Neste ambiente favorável ao desenvolvimento de novas tecnologias, surgiram diversas aplicações nos últimos anos, como reconhecimento facial, detecção de recursos, reconstrução 3D e fotografia computacional (SZELISKI, 2022).

1.2.1 Técnicas Clássicas

No âmbito de visão computacional, as técnicas clássicas dizem respeito a um conjunto de teorias e abordagens matemáticas e lógicas capazes de solucionar problemas. Dentre as técnicas clássicas mais utilizadas em aplicações de visão computacional, se destacam:

- **Detecção de características:** com o auxílio de transformadas, filtros e algoritmos, busca-se identificar, em uma imagem, alguns pontos de interesse, como bordas, cantos, *key-points*, etc;
- **Detecção de movimento:** utilização de algoritmos como Lucas-Kanade e Horn-Schunk para detectar o movimento em uma sequência de *frames*;
- **Segmentação de imagens:** separação de objetos ou regiões relevantas na imagem. Alguns dos algortimos mais importantes são o método de *watershed* e o algoritmo de segmentação por agrupamento.

1.2.2 Heurísticas

Heurísticas são regras e observações mais simples que buscam elaborar uma solução direta, porém imperfeita e aproximada, para problemas complexos. Vamos supor que seja de nosso interesse a elaboração de um sistema de reconstrução 3D a partir de múltiplas imagens 2D. Um exemplo de heurística é assumir que as câmeras estão posicionadas radialmente ao redor do centro do objeto ou região de interesse. Assim, o problema torna-se menos complexo, apesar de ter uma solução aproximada. Outro exemplo de heurística em visão computacional é limiarização, ou, do inglês, *thresholding*. Essa técnica serve para realizar a binariação de uma imagem em escalas de cinza com a utilização de um "gatilho", ou *threshold*. Para cada pixel, se o mesmo for maior que o *threshold*, receberá preto. Do contrário, será branco.

Sinteticamente, heurísticas fornecem soluções rápidas, não tão precisas e de fácil implementação. As técnicas clássicas trazem soluções mais complexas e bem elaboradas. Tanto heurísticas quanto técnicas clássicas podem ser utilizadas concomitantemente em um

mesmo sistema, abstraindo características importantes de cada abordagem com o fito de tentar obter o melhor resultado de cada e, consequentemente, do sistema.

1.3 Inteligência Artificial

Nem todos os autores concordam sobre uma definição única para AI. Existem duas principais vertentes de definições: as definições que se preocupam em relacionar processos de pensamento e raciocínio; e as que abordam questões comportamentais. Entretanto, uma definição prática é dizer que AI é um sistema computacional capaz de realizar tarefas que necessitam do raciocínio humano, como Processamento de Linguagem Natural (NLP, na sigla em inglês), reconhecimento de padrões e tomada de decisões (RUSSELL et al., 2010).

Um dos principais ramos da IA é o ML. Tal área se concentra na pesquisa e desenvolvimento de algoritmos que permitam que sistemas inteligentes aprendam com dados. O objetivo é criar modelos matemáticos treináveis que possam realizar tarefas específicas, como classificações e tomadas de decisão. Alguns dos principais tipos de aprendizado de máquina são:

- **Aprendizado supervisionado:** o modelo é treinado a partir de um conjunto de dados rotulados com o objetivo de diferenciar classes em dados não-rotulados;
- **Aprendizado não-supervisionado:** o modelo é treinado a partir de dados não-rotulados com o objetivo de permitir que o modelo reconheça padrões e diferencie os dados;
- **Aprendizado por reforço:** o modelo é treinado para receber "recompensas"a cada decisão correta, estabelecendo uma relação diretamente proporcional entre acertos e recompensa.

1.3.1 Aprendizado Profundo

O Aprendizado Profundo, ou *Deep Learning*, é uma subárea do ML que busca imitar a estrutura e funcionalidade do cérebro humano através da construção de modelos complexos e hierárquicos de processamento. Para isso, são utilizadas Redes Neurais Artificiais (ANNs, da sigla em inglês) profundas com várias camadas intermediárias de processamento, buscando melhor evoluir e otimizar a acurácia. (MOHIT DAYAL MEHAK GUPTA, 2023) ANNs (HAYKIN, 2001) são hoje o artifício tecnológico que mais se assemelha ao funcionamento de um cérebro humano. Da mesma forma que o cérebro humano depende do funcionamento conjunto de sua unidade funcional, o neurônio, as ANNs operam conforme várias camadas de neurônios artificiais interligadas, onde a entrada de cada camada relaciona-se da saída da camada anterior. A primeira camada é chamada de camada de entrada, a última é chamada

de camada de saída, e as camadas intermediárias são chamadas de camadas ocultas. Cada neurônio de uma ANN calcula a soma ponderada de todas as entradas, aplica uma função de ativação a essa soma envia o valor na sequência. Há também um fator de viés, chamado de *bias*, que entra no cálculo da soma independentemente de qualquer peso, evitando que o valor computado seja igual a zero. Cada entrada do neurônio possui a ela um peso associado, cuja função é dar importância e estabelecer hierarquia aos sinais de entrada. Uma ilustração de rede neural encontra-se na Figura 1.1

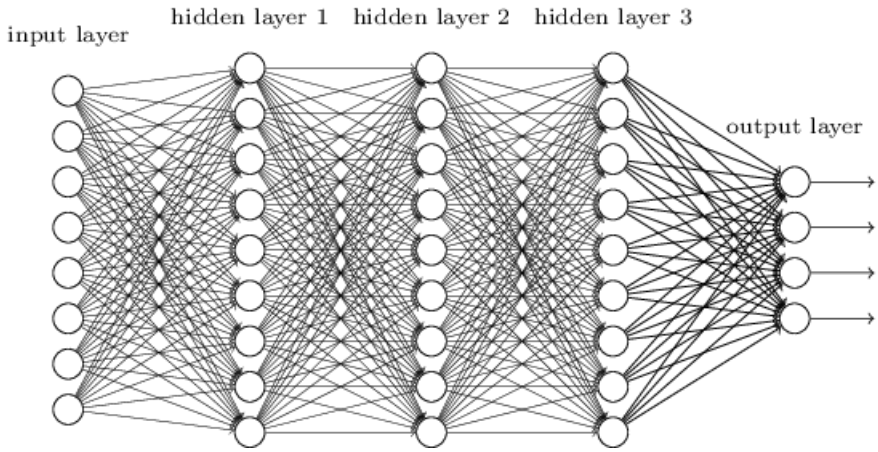


Figura 1.1 – Rede Neural Artificial

Fonte: Melo (2019)

A função de ativação aplica um fator de não-linearidade ao processo, permitindo que a rede neural se adéque a problemas mais complexos, visto que a simples computação das entradas e os pesos não seriam suficientemente eficazes para a realização de tarefas mais abstratas, como reconhecimento de padrões e tradução, por exemplo. Em tese, há dois principais desafios ao se criar uma ANN eficaz: definir sua estrutura e calcular o conjunto de pesos que provê maior acurácia ao modelo. Quanto ao primeiro processo, uma análise qualitativa do projeto consegue traçar um bom caminho sobre quais modelos de ANN tentar; a depender do tipo de tarefa a ser executada, há uma arquitetura de ANN que melhor se ajuste. Quanto ao segundo, os pesos são atribuídos e atualizados na fase de treinamento. A cada par entrada-saída, um erro é obtido a partir da diferença entre a saída esperada e a saída efetiva, o qual influenciará no cálculo de ajuste dos pesos. Assim, após sucessivas computações, a ANN se ajusta iterativamente aos dados.

Apesar de serem ferramentas poderosas para tarefas complexas, as ANNs podem ser computacionalmente exigentes e necessitam de grandes volumes de dados para um treinamento eficaz.Além disso, a interpretabilidade dos resultados e a evolução do modelo podem não ser tão claras (SO et al., 2020).

1.4 Objetivos

A proposta deste trabalho é desenvolver e comparar diferentes arquiteturas e *frameworks* de redes neurais, como YOLO (Darknet, Ultralytics e Meituan) e Detectron2 (RetinaNet e Faster R-CNN), utilizados na detecção de objetos em imagens para identificar linhas de plantação em imagens aéreas agrícolas obtidas por drones e, a partir disso, extrair informações adicionais como falhas e a porcentagem da área usada para o cultivo das plantas. Além disso, serão utilizadas técnicas de *data augmentation* e *synthetic data* para aumentar nosso banco de dados a ser utilizado no treinamento das redes neurais.

1.4.1 Objetivo Geral

Detectar linhas de plantação, identificar falhas e calcular a porcentagem do solo usado para o cultivo em imagens aéreas agrícolas adquiridas por meio de drone.

1.4.2 Objetivos Específicos

- Utilizar *data augmentation* e *synthetic data* para aumentar o banco de dados;
- Rotular manualmente as imagens do banco de dados.
- Implementar, treinar e testar diferentes arquiteturas e *frameworks* de redes neurais para detectar as linhas de plantação;
- Analisar e comparar o desempenho das redes neurais utilizadas;
- Desenvolver algoritmos para identificar as falhas e calcular a porcentagem do solo cultivada.

1.5 Estrutura do Texto

Este texto está organizado da seguinte forma: na seção 2 serão detalhadamente discutidos os conceitos por trás das Redes Neurais Convolucionais e das diversas arquiteturas de rede utilizadas, revelando suas complexidades e funcionalidades. Na seção 3, é realizada uma revisão de literatura sobre o uso de drones na agricultura, além de técnicas clássicas e de *deep learning* em imagens aéreas para extrair informações como linhas e presença de ervas daninhas nas plantações. Na seção 4, apresenta-se a metodologia aplicada no trabalho em relação a base de dados utilizada, a forma como foram realizados os treinamentos e os testes das diferentes arquiteturas, assim como os algoritmos desenvolvidos para identificar as falhas e calcular a porcentagem de área cultivada. Na seção 5, são exibidos os resultados que foram obtidos através da implementação das arquiteturas em nossa base de dados. Por fim, na seção

6, discutimos os resultados encontrados, as dificuldades encontradas e sugerimos trabalhos futuros que podem ser desenvolvidos utilizando as técnicas abordadas neste estudo.

2 Fundamentação Teórica

2.1 Redes Neurais Convolucionais

Redes Neurais Convolucionais (CNNs, na sigla em inglês) são um tipo específico de ANNs amplamente utilizadas no campo da visão computacional, especialmente em tarefas como detecção e segmentação de objetos, reconhecimento de padrões, entre outras. A principal característica das CNNs é o processamento em quatro camadas principais:

- Camada de convolução;
- Camada de *pooling*;
- Camada de *flattening*;
- Rede neural densa ou camada totalmente conectada.

A camada de convolução constitui a base do funcionamento de uma CNN. Nessa camada, aplica-se um conjunto de filtros, ou *kernels*, que são matrizes de pesos aplicadas à imagem. O filtro multiplica o peso pelo pixel correspondente e soma os resultados. Em seguida, o filtro é deslocado em s pixels, onde s é chamado de passo ou *stride*. Esse processo continua até que o filtro percorra toda a imagem, tanto na vertical quanto na horizontal, gerando uma matriz reduzida chamada mapa de características ou *feature map*, que contém apenas as informações extraídas pelos filtros. A convolução de uma matriz de pixels I, de dimensão $m \times n$, com um *kernel* K pode ser representada matematicamente pela equação (2.1).

$$S[i,j] = (I * K)[i,j] = \sum_m \sum_n I[i-m, j-n]K[m,n] \tag{2.1}$$

Onde $S[i,j]$ se refere ao elemento na posição $[i,j]$ do mapa de características. O treinamento de uma CNN ocorre pelo ajuste dos pesos de cada *kernel* do conjunto utilizado, otimizando a busca pelas características mais relevantes. A Figura 2.3 exemplifica a aplicação de um *kernel* (Figura 2.2) de tamanho 3×3 em uma imagem de tamanho 5×5.

Figura 2.2 – Exemplo de *kernel*

Fonte: Dumoulin e Visin (2016, p. 6)

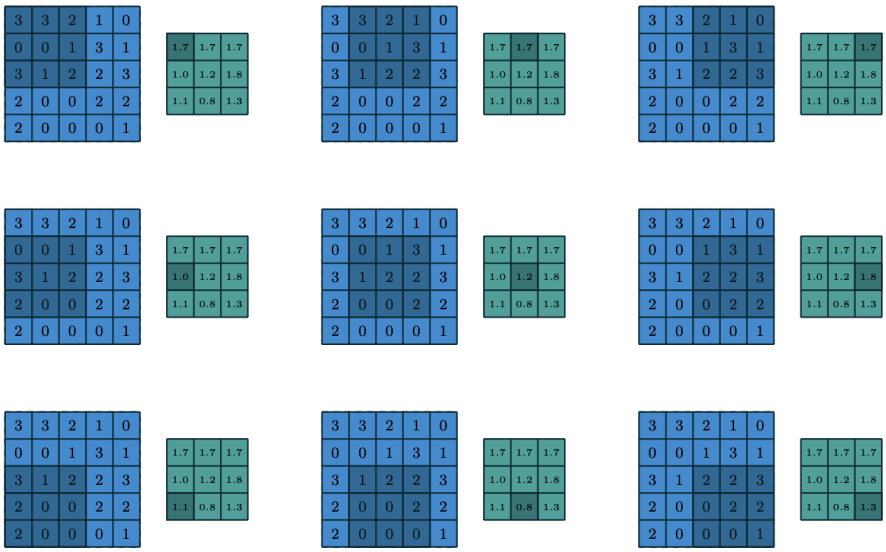


Figura 2.3 – Exemplo de convolução

Fonte: Dumoulin e Visin (2016, p. 7)

A convolução gera *d feature maps* de dimensões reduzidas em relação à imagem de entrada, onde d é a quantidade de filtros utilizados na camada de convolução. As dimensões reduzidas de um *feature map* podem ser representadas pela equação (2.2):

$$x = \frac{m-k}{s} + 1 \qquad\qquad y = \frac{n-k}{s} + 1 \tag{2.2}$$

Onde k é o tamanho do *kernel*, $m \times n$ são as dimensões da imagem original, s é o *stride* e $x \times y$ são as dimensões do *feature map*. Uma maneira de manter a dimensão original da imagem é utilizando a técnica de *zero-padding*, que consiste em criar uma borda de zeros ao redor da matriz original.

Após a obtenção dos *feature maps*, realiza-se uma subamostragem na camada de *pooling*. De maneira análoga à etapa de convolução, a camada de *pooling* aplica um filtro que desliza pelos *feature maps* e gera como resultado um único pixel a partir dos elementos sobrepostos pelo filtro. Existem algumas maneiras de definir o valor do pixel, como:

- ***MaxPooling*:** escolha do maior pixel;
- ***SumPooling*:** soma dos pixels;
- ***AveragePooling*:** média dos pixels.

O objetivo desse processo é reduzir o tamanho do mapa de características, que muitas vezes contém informações desnecessárias e redundantes. A Figura 2.4 ilustra o procedimento.

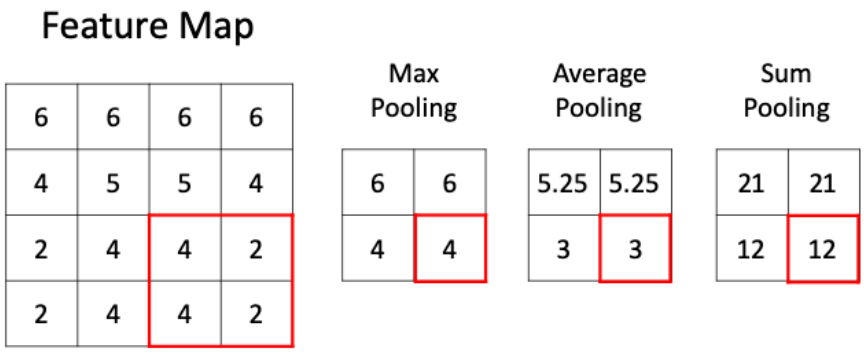


Figura 2.4 – Tipos de *pooling*

Fonte: Hussain (2020)

Posteriormente, as matrizes resultantes são enviadas à camada de *flattening*, cuja função é redimensionar a matriz para que ela se torne um vetor unidimensional, como ilustrado na Figura 2.5.

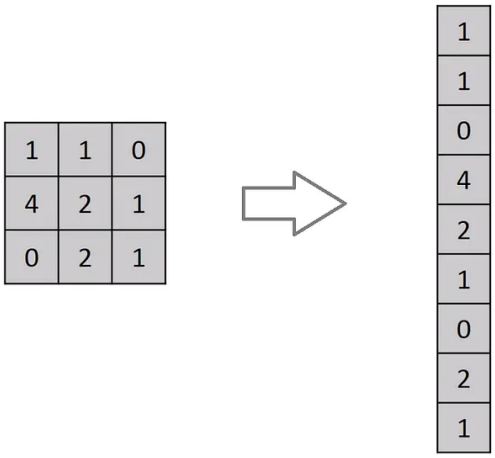


Figura 2.5 – *Flattening*

Fonte: Hussain (2020)

Em seguida, cada elemento do vetor resultante alimenta os neurônios da camada de entrada da rede neural densa, cujo objetivo é encontrar os melhores pesos para gerar a saída desejada. Vale ressaltar que define-se por saída desejada uma classificação correta.

A arquitetura completa e os processos envolvidos podem ser resumidos pela Figura 2.6.

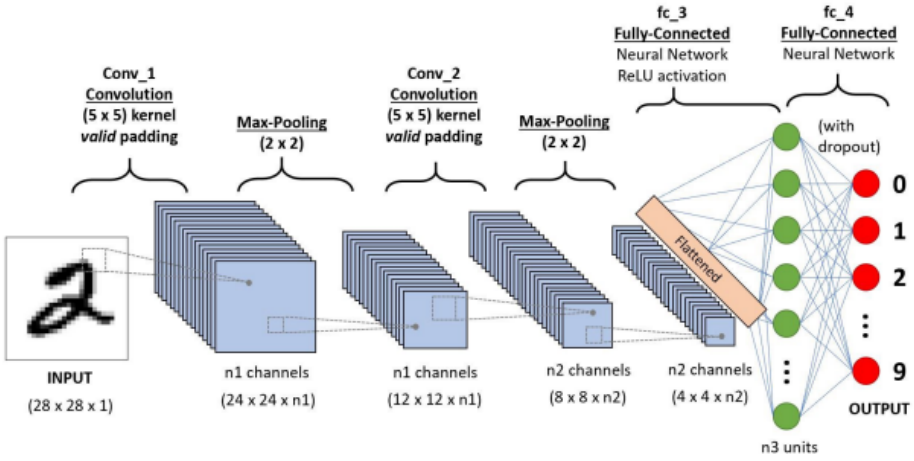


Figura 2.6 – Arquitetura CNN

Fonte: Meghakumar (2021)

A função de ativação comumente empregada em CNNs é a ***Rectified Linear Unit*** (ReLU). A ReLU ajuda a decidir quando um neurônio deve disparar sua saída ou não. Basicamente, caso o valor do neurônio seja negativo, a saída será zero; caso seja um valor positivo, a ReLU funciona de maneira idêntica à função de ativação linear, ou seja, a saída será o próprio valor do neurônio. Formalmente, a ReLU é descrita pela equação (2.3).

$$R(z) = max(0, z) \tag{2.3}$$

Onde z é o valor do neurônio. Outra função de ativação frequentemente utilizada em CNNs é a função *Softmax*. A função *Softmax* é especialmente útil na camada de saída de redes neurais para problemas de classificação, uma vez que ela converte a saída em probabilidades, facilitando a interpretação dos resultados. A equação (2.4) descreve a função *Softmax*.

$$S(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{G} e^{z_j}} \tag{2.4}$$

Onde z_i é a saída do neurônio i, e G é o número de neurônios na camada de saída.

O treinamento de uma CNN geralmente utiliza o algoritmo de otimização chamado *BackPropagation* (KOECH, s.d.) em conjunto com uma função de perda, como a entropia cruzada (*Cross-Entropy*), que mede a diferença entre as saídas desejadas e as saídas geradas pela rede.

2.2 Arquiteturas de Detecção de Objetos

2.2.1 Faster R-CNN

Para entendermos melhor o funcionamento da Faster R-CNN, faz-se necessária uma breve introdução a respeito de suas antecessoras: a *Region-based Convolutional Neural Network* (**R-CNN**) e a *Fast Region-based Convolution Neural Network* (**Fast R-CNN**).

Um *pipeline* básico de detecção de objetos consiste na sucessão das seguintes operações:

- **Propositor de regiões:** a partir de uma imagem de entrada, sugere regiões onde um objeto pode ou não estar, utilizando algoritmos como *Selective Search*;
- **Extrator de características:** este componente, para cada região proposta da imagem, extrai um vetor de características utilizando algoritmos como o Histograma de Gradientes Orientados (HOG, da sigla em inglês);
- **Algoritmo classificador:** classifica o vetor de características utilizando técnicas como *Support Vector Machine* (SVM), indicando se existe ou não a presença de algum objeto de interesse na região proposta.

Nesse sentido, a R-CNN, arquitetura de múltiplos estágios desenvolvida em 2014 por um grupo de pesquisadores da Universidade de Califórnia, trouxe um diferencial ao *pipeline* tradicional: a utilização de uma CNN para extrair as características na etapa 2 (vide Figura 2.7). Entretanto, a R-CNN possui alguns contrapontos, como a vagarosidade do *Selective Search* para propor as regiões, a alta requisição de memória, uma vez que as características extraídas são armazenadas em *cache* para posterior treinamento do SVM. Além disso, a arquitetura não permite a detecção em tempo real, visto que cada região proposta é processada separadamente pelo extrator de características. Além disso, como é composta por três módulos independentes, essa arquitetura dificulta o treinamento *end-to-end* (ponta a ponta).

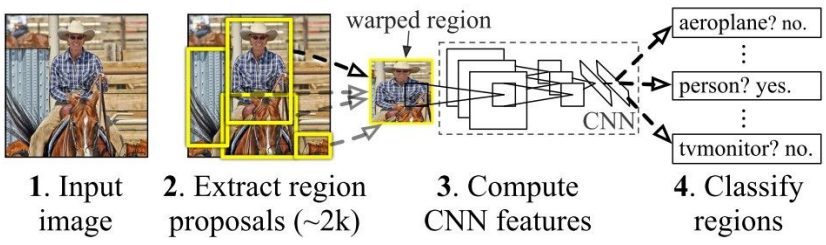


Figura 2.7 – Arquitetura da R-CNN

Fonte: (GAD, 2020)

Assim sendo, Ross Girshick, pesquisador da *Facebook AI Research* (FAIR), desenvolveu a *Faster R-CNN*, uma arquitetura que combatia alguns dos contrapontos supracitados. A implementação dela trouxe maior velocidade, acurácia e menor requisição de memória do que sua antecessora. Essas melhorias são explicadas pelas seguintes incrementações:

- **Camada de *pooling* adapativo de regiões:** responsável por gerar vetores de características de tamanho igual para todas as regiões propostas (independentemente da forma da região), facilitando a representatividade das regiões;
- **Arquitetura de estágio único:** recebe a imagem e já dá como resposta as caixas delimitadoras e suas respectivas probabilidades de conter objetos;
- **Processamento conjunto das regiões propostas:** possibilitado pela inclusão da nova camada de *pooling*;
- **Menor requisição de memória:** devido à não utilização da *cache* para armazenar os vetores de características.

Essa arquitetura é ilustrada na Figura 2.8.

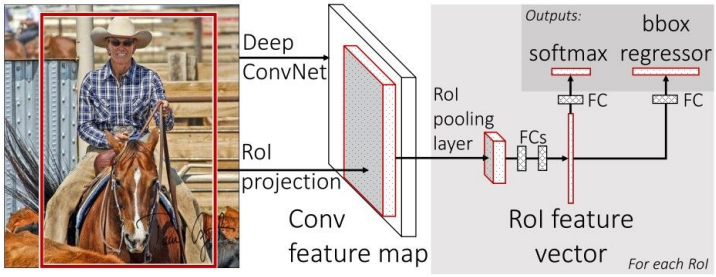


Figura 2.8 – Arquitetura da Fast R-CNN

Fonte: (GAD, 2020)

As regiões propostas pelo *Selective Search* são processadas pela CNN, gerando um mapa de características. Em seguida, cada mapa passa pela camada *pooling* de regiões, gerando vetores de mesmo tamanho para todas as regiões. Posteriormente, o vetor alimenta duas redes densas completamente conectadas: uma para regressão da caixa delimitadora e outra para a classificação probabilísticas de a região conter ou não um objeto, utilizando a função de ativação *softmax*.

Apesar das melhorias apresentadas, o *Selective Search* ainda consome um tempo computacional considerável para propor as regiões. Nesse viés, a *Faster R-CNN* (REN et al., 2016), modelo de dois estágios desenvolvido em 2015 por um grupo de pesquisadores da Microsoft, trouxe como alternativa a incrementação da *Region Proposal Network* (RPN), CNN cuja funcionalidade é substituir o *Selective Search* na proposição de regiões, à estrutura da *Fast R-CNN*. Além disso, os cálculos convolucionais realizados na RPN e na *Fast R-CNN*

são compartilhados, o que aumenta a eficiência do tempo de processamento. A arquitetura é ilustrada na Figura 2.9.

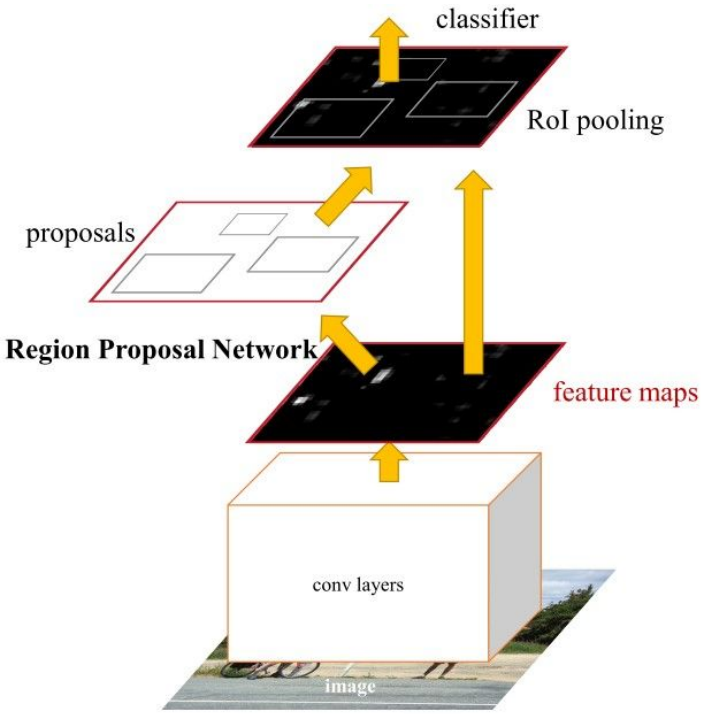


Figura 2.9 – Arquitetura da Faster R-CNN

Fonte: Ren et al. (2016, p. 3)

Não obstante, os desenvolvedores da arquitetura introduzem a ideia de âncoras, que nada mais são do que caixas delimitadores de referência com escala e proporção bem especificados. Isso permite que uma mesma região possua representações com escalas e proporções diferentes, possibilitando detecções sob diferentes perspectivas.

Após o recebimento da imagem, a RPN propõe regiões candidatas a conter um objeto. Para cada região, é gerado um vetor de características de tamanho fixo através do processamento da camada de *Pooling* de regiões, os quais são então classificados seguindo o mesmo procedimento utilizado na *Fast R-CNN*. Finalmente, o sistema retorna caixas delimitadoras acompanhadas da respectiva probabilidade de cada uma conter um objeto.. (GAD, 2020)

2.2.2 RetinaNet

Outra arquitetura é a RetinaNet, cuja principal contribuição é a utilização da função *Focal Loss* (LIN, T.-Y. et al., 2018). Essa função surge do desbalanceamento de classes (vide Figura 2.10) em arquiteturas de estágio único, as quais são, em geral, piores do que arquiteturas de dois estágios.

Tradicionalmente, a função de perda utilizada em modelos de classificação é a *Cross Entropy* (CE), formalizada na equação (2.5).

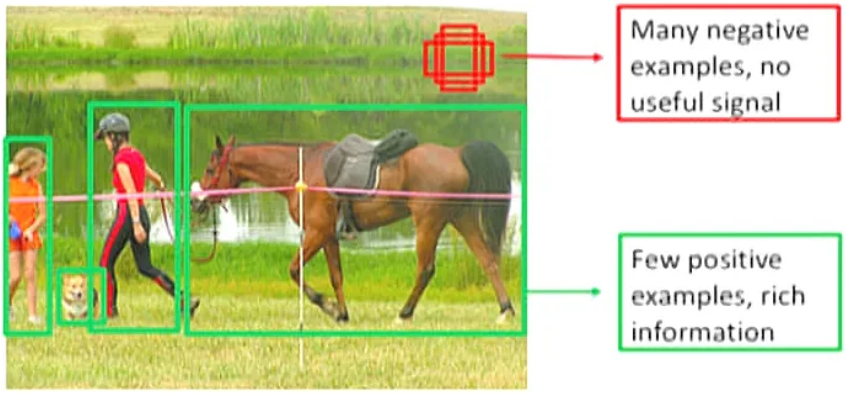


Figura 2.10 – Desbalanceamento de classes

Fonte: (TSANG, 2019)

$$CE(Y_i, p_i) = -\sum_{i=1}^{i=n} Y_i \log_b(p_i) \tag{2.5}$$

Nesta equação, Y representa a classe verdadeira e p_i a probabilidade estimada. Essa função não lida bem com desbalanceamento de classe, visto que a grande quantidade de regiões de fácil classificação predominam sobre a baixa quantidade de regiões de difícil classificação, enviesando a descida do gradiente e a função de perda, ajustando os pesos numa direção que o torna mais confiante para previsões fáceis e menos para previsões difíceis (TSANG, 2019). Além disso, o sistema falha discernir exemplos onde ele comete erros grosseiros e exemplos de fácil classificação.

Uma alternativa para o problema do desbalanceamento é a *Balanced Cross-Entropy loss*, que adiciona um fator de balanceamento α à cada classe no sistema, com magnitude inversamente proporcional à frequência com que a classe aparece. Formalmente, a *Balanced Cross-Entropy loss* pode ser definida como na equação (2.6).

$$BCE(Y_i, p_i) = -\sum_{i=1}^{i=n} \alpha_i Y_i \log_b(p_i) \tag{2.6}$$

Entretanto, o problema da diferenciação de classificações fáceis e difíceis ainda persiste. Nesse sentido, o *Focal loss* é uma função de perda que consegue atribuir mais foco aos exemplos onde costuma errar grosseiramente. Assim, a função CE é ponderada de modo a atribuir mais foco nos exemplos de difícil classificação, como segue na equação (2.7).

$$FL(Y_i, p_i) = -\sum_{i=1}^{i=n} (1 - p_i)^{\gamma} \log_b(p_i) \tag{2.7}$$

γ é um parâmetro otimizável, chamado de parâmetro de foco. Existem duas principais vertentes de ação do parâmetro de foco:

- Quando o sistema erra a classificação, p_i é pequeno, o que faz $(1-p_i)^\gamma \to 1$, fazendo a função se comportar como a *CE*;
- Para classificações muito fáceis, $p_i \to 1$, e, consequentemente $(1-p_i)^\gamma \to 0$, diminuindo o impacto desses exemplos na função de perda.

Além da *Focal loss*, a RetinaNet (vide Figura 2.11) implementa um novo elemento importante para a detecção de objetos: a *Feature Pyramid Network* (FPN). A FPN é capaz de extrair *features* de diferentes proporções e tamanhos por meio de operações convolucionais, empilhando os *feature maps* para posterior processamento.

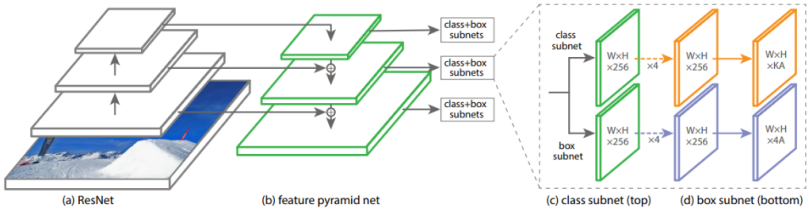


Figura 2.11 – Arquitetura da RetinaNet

Fonte: Tsung-Yi Lin et al. (2018, p. 5)

A FPN é utilizada em conjunto com uma CNN profunda, como ResNet ou *Visual Geometry Group* (VGG), que são responsáveis pela extração profunda de características. Em seguida, cada camada da FPN é então processada de maneira análoga à *Faster R-CNN* por duas redes neurais densas subsequentes: a de classificação e a de regressão das caixas delimitadoras. É importante frisar que a rede de classificação atribui uma probabilidade para cada âncora presente em cada camada. Já a rede de regressão das caixas delimitadoras é responsável por calcular o desvio entre a posição da âncora e a posição real de um objeto dentro de uma caixa delimitadora previamente estabelecida.

2.2.3 YOLO

A arquitetura de redes neurais *You Only Look Once* (YOLO) é uma abordagem inovadora no campo da detecção de objetos em tempo real. Ela revolucionou a maneira de tratar a detecção de objetos por não seguir os métodos tradicionais baseados em *region proposal networks*, como a *Faster R-CNN*, e conseguir atingir alta precisão e velocidade no processamento das imagens. A YOLO foi inicialmente proposta por Joseph Redmon (REDMON; DIVVALA et al., 2016), desenvolvedor do Darknet, um *framework* de aprendizado profundo de código aberto escrito em C e CUDA (REDMON, 2013–2016), utilizado como base da YOLO devido a sua eficiência e facilidade de uso. A ideia central por trás desta arquitetura é processar a imagem como um todo. Para isso, a imagem de entrada é dividida em uma grade e, em cada célula da grade, é atribuída a tarefa de prever *bounding boxes* com a respectiva

probabilidade de se ter algum objeto dentro dela, além da probabilidade do objeto pertencer a alguma classe específica. As Figuras 2.12 e 2.13 mostram o funcionamento desta arquitetura.

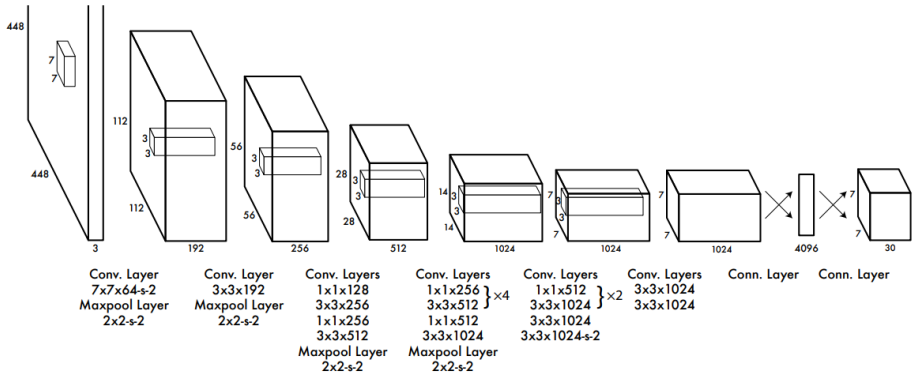


Figura 2.12 – Arquitetura original da YOLO

Fonte: Redmon, Divvala et al. (2016, p. 3)

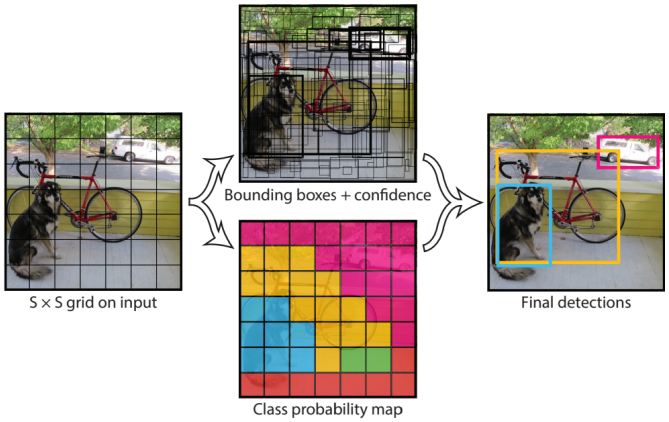


Figura 2.13 – Funcionamento da YOLO (divisão da imagem em grade e previsões das *bounding boxes*)

Fonte: Redmon, Divvala et al. (2016, p. 2)

Desde a primeira versão, a YOLO passou por várias melhorias e otimizações, chegando a sua terceira versão, YOLOv3 (REDMON; FARHADI, 2018), a qual apresenta detecção em várias escalas para identificar objetos pequenos. Ela utiliza a arquitetura Darknet-53 para melhor desempenho e adota *bounding boxes* ajustadas com base em âncoras para melhor adaptação aos formatos dos objetos. Outra versão também baseada no *framework* Darknet, criada por Alexey Bochkovskiy em colaboração com Chien-Yao Wang e Hong-Yuan Mark Liao, é a YOLOv4 (BOCHKOVSKIY; WANG; LIAO, 2020), que utiliza uma arquitetura mais eficiente (CSPDarknet53) e os módulos *Path Aggregation Network* (PANet) e *Spatial Pyramid Pooling* (SPP) para uma melhor detecção de objetos, possui um melhor desempenho graças

à ativação *Mish* e às técnicas de otimização. Para visualizar melhor onde se encontra a diferença entre as arquiteturas da YOLOv3 e YOLOv4, a Figura 2.14 mostra uma arquitetura típica de um modelo de detecção de objeto, a qual possui:

- ***Backbone*:** responsável pela extração de características da imagem. Na YOLOv3, o *backbone* utilizado é o Darknet-53, enquanto na YOLOv4, temos a introdução do CSPDarknet53, que combina a arquitetura Darknet-53 com a técnica *Cross-Stage-Partial-connections* (CSP), resultando em melhor eficiência e desempenho;
- ***Neck*:** é a parte intermediária da rede que ajuda a refinar as características extraídas e a conectar diferentes níveis de informações. Na YOLOv3, nessa parte é usada a FPN. Já na YOLOv4, são incorporados dois módulos adicionais: PANet e SPP, que melhoram a detecção de objetos em diferentes escalas e contextos;
- ***Head*:** é a camada final da rede, responsável por gerar as previsões de *bounding boxes* e classificações de objetos. Tanto na YOLOv3 quanto na YOLOv4, o *head* utiliza uma abordagem similar, mas na YOLOv4, temos melhorias nas técnicas de ativação e normalização, como a adoção da função de ativação *Mish* e o método de normalização *Cross mini-Batch Normalization*.

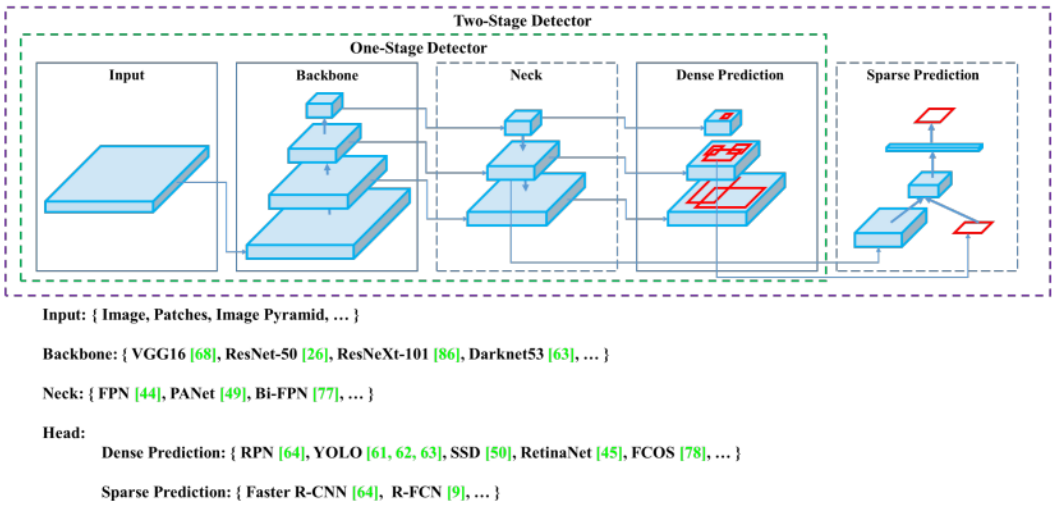


Figura 2.14 – Arquitetura típica de um modelo de detecção de objeto

Fonte: Bochkovskiy, Wang e Liao (2020, p. 2)

A versão seguinte, YOLOv5, desenvolvida pela empresa Ultralytics (JOCHER et al., 2022), foi construída usando PyTorch, um *framework* popular e amplamente utilizado para desenvolvimento de aprendizado profundo. Essa mudança facilita a integração com outros projetos e ferramentas baseados em PyTorch e simplifica o treinamento, a avaliação e a

implantação dos modelos YOLOv5. Outras melhorias em relação à versão anterior se dão nos seguintes aspectos:

- **Arquitetura simplificada:** a YOLOv5 possui uma arquitetura mais enxuta e modular, facilitando a personalização e a adaptação às necessidades específicas de diferentes aplicações;
- **Modelos com diferentes tamanhos:** a YOLOv5 oferece cinco variantes de tamanho (YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l e YOLOv5x), permitindo aos usuários escolher entre diferentes *trade-offs* de precisão e velocidade, dependendo dos requisitos do projeto;
- **Melhorias no treinamento:** a YOLOv5 apresenta novas técnicas de aumento de dados, como o *MixUp*, que melhora a robustez e a generalização do modelo;
- **Suporte para exportação de modelos:** a YOLOv5 suporta a exportação de modelos treinados para diversos formatos, como ONNX, TensorFlow e Core ML, facilitando a implantação em várias plataformas.

A próxima versão cronológica da YOLO corresponde à YOLOv7, cujo código-fonte é de autoria de Wong Kin-Yiu (WONG, 2021) e é implementada originalmente utilizando o *framework* PyTorch. No entanto, a análise mais detalhada sobre a arquitetura é feita no artigo Wang, Bochkovskiy e Liao (2022), de autoria dos mesmos criadores da YOLOv4. Neste trabalho, foi utilizada a versão da YOLOv7 adaptada por Alexey Bochkovskiy para o *framework* Darknet (ALEXEYAB, 2021), a qual mantém as principais melhorias e inovações da arquitetura original da YOLOv7.A adaptação também permite que os usuários aproveitem as vantagens da arquitetura YOLOv7, ao mesmo tempo em que se beneficiam do desempenho e da eficiência do Darknet.

A YOLOv7 possui como principais características e melhorias na arquitetura e na performance:

- **E-ELAN (*Extended Efficient Layer Aggregation Network*):** é o bloco computacional presente no *backbone* da YOLOv7 que visa melhorar a eficiência e a extração de características de diferentes níveis. Inspirado em pesquisas anteriores sobre eficiência de redes, o E-ELAN utiliza a expansão, embaralhamento e fusão de cardinalidade (XIE et al., 2017), a qual corresponde a quantidade de transformações que são aplicadas aos dados de entrada, para melhorar continuamente a capacidade de aprendizado da rede sem interromper o caminho original do gradiente;
- ***Compound Model Scaling*:** é usado para escalonar o modelo da YOLOv7 considerando três dimensões principais: resolução (tamanho da imagem de entrada), largura

(número de canais) e profundidade (número de camadas). Em outros modelos de detecção de objetos, o escalonamento é feito de forma independente em cada uma dessas dimensões. No entanto, o *Compound Model Scaling* propõe um escalonamento coordenado, levando em consideração o impacto combinado das três dimensões no desempenho e na complexidade do modelo. Isso é feito encontrando um equilíbrio adequado entre resolução, largura e profundidade, permitindo que o modelo cresça de forma otimizada. (TAN; LE, 2020);

- ***Trainable bag-of-freebies*:** conjunto de técnicas que melhoram o desempenho do modelo sem aumentar o custo de treinamento (ZHANG et al., 2019). No contexto da YOLOv7, algumas técnicas são:
  - ***Planned Re-parameterized Convolution (RepConv)*:** é uma técnica usada após o treinamento para melhorar os resultados de inferência do modelo. Na YOLOv7, o uso do RepConvN (*RepConv without identity connection*) para projetar a arquitetura da convolução planejada re-parametrizada, garante que não haja conexão de identidade quando uma camada convolucional com resíduo ou concatenação é substituída pela convolução re-parametrizada;
  - ***Coarse for auxiliary and fine for lead loss*:** é um conceito que consiste em acrescentar uma *Auxiliary Head* para auxiliar no treinamento nas camadas intermediárias. Isso permite a Supervisão Profunda (*Deep Supervision*), em que o *label assigner* (mecanismo que considera os resultados de previsão da rede juntamente com o *ground truth*) atribui rótulos grosseiros (*coarse*) para a *Auxiliary Head* e rótulos refinados (*fine*) para a *Lead Head*.

Outra versão da YOLO, desenvolvida quase em paralelo com a YOLOv7, foi a YOLOv6, lançada pouco tempo depois pela Meituan (LI et al., 2022), que possui como foco principal atender às demandas das aplicações industriais, oferecendo desempenho adaptável a uma variedade de *hardwares* e situações reais. Considerando a importância da velocidade e precisão, a versão apresenta cinco modelos distintos (*Nano*, *Tiny*, *Small*, *Medium*, *Large*).

Diferentemente das versões anteriores da YOLO, que utilizam métodos baseados em âncoras para detecção de objetos, a YOLOv6 adota uma abordagem livre de âncoras. Algumas outras mudanças na arquitetura são:

- ***Backbone*:** utilização da EfficientRep, a qual é composta por blocos RepBlock, RepConv e CSPStackRep, que são usados em diferentes tamanhos da YOLOv6, como *Nano*, *Tiny*, *Small*, *Medium* e *Large*. Esse *backbone* reparametrizado foi desenvolvido para melhorar o equilíbrio entre precisão e velocidade durante a inferência. Nas versões *Nano*, *Tiny* e *Small* da YOLOv6, a EfficientRep utiliza redes VGG reparametrizadas.. Como observado na Figura 2.15, durante a fase de treinamento, o *backbone* apresenta

Skip Connections no bloco RepBlock, enquanto durante a inferência essas conexões são substituídas por blocos convolucionais 3x3 (RepConv) mais simples e rápidos. Para os modelos *Medium* e *Large*, o *backbone* EfficientRep é baseado em versões reparametrizadas do *backbone* CSP, chamadas CSPStackRep. Essa abordagem garante um melhor desempenho tanto em termos de precisão quanto de velocidade, atendendo às demandas de aplicações em tempo real e industriais;

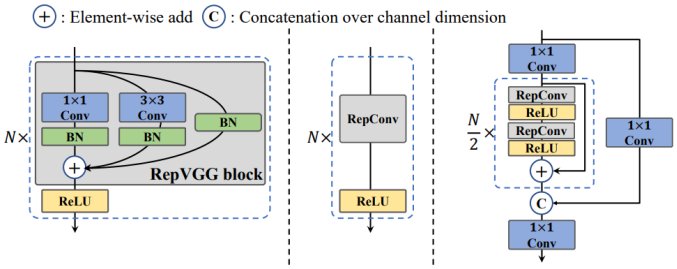


Figura 2.15 – Representação visual do RepBlock, RepConv e CSPStackRep, respectivamente

Fonte: Li et al. (2022, p. 4)

- ***Neck*:** é utilizado a rede PAN assim como nas demais versões da YOLO, porém no caso da YOLOv6 ela concatena recursos de diferentes blocos reparametrizados e, portanto, é chamado de PAN reparametrizado (Rep-PAN);
- ***Head*:** usa-se *Efficient Decoupled Head*, na qual os ramos de classificação e detecção não compartilham parâmetros e se ramificam separadamente a partir do *backbone*. Essa estratégia reduz ainda mais os cálculos e melhora a precisão;
- ***Loss Functions*:** há a presença de duas funções de perda, a *Varifocal Loss* (VFL), responsável por classificação e a *Distribution Focal Loss* (DFL), usada nos modelos *Medium* e *Large*, a qual trata a distribuição contínua das localizações das *bounding boxes* como uma distribuição de probabilidade discretizada.

Atualmente, a última versão da YOLO é a YOLOv8, desenvolvida pela Ultralytics, que é um modelo de ponta e de última geração que se baseia no sucesso das versões anteriores, apresentando novos recursos e aprimoramentos para melhorar o desempenho, flexibilidade e eficiência. Projetada para ser rápida, precisa e fácil de usar, a YOLOv8 é uma excelente opção para uma ampla gama de tarefas de detecção e rastreamento de objetos, segmentação de instâncias, classificação de imagens e estimativa de pose (ULTRALYTICS, 2023).

Este novo modelo apresenta uma API amigável (acessada diretamente por linha de comando ou usando Python), maior velocidade e precisão. Ele possui uma nova rede *backbone*, uma *Head* sem âncoras e uma nova função de *Loss*. Além disso, a YOLOv8 oferece suporte a vários formatos de exportação e pode ser executado em CPUs e GPUs. Assim como

a versão anterior desenvolvida também pela Ultralytics, este modelo oferece cinco modelos para detecção, segmentação e classificação. Os modelos variam de *Nano* (mais rápido e menor) a *Extra Large* (mais preciso, porém mais lento). Os modelos pré-treinados incluem detecção de objetos no *COCO detection dataset* (resolução da imagem de 640), segmentação de instâncias no *COCO segmantation dataset* (LIN, T. et al., 2014) (resolução da imagem de 640) e classificação de imagens no *ImageNet* (resolução da imagem de 224) (DENG et al., 2009).

Até o momento, não foi publicado o artigo detalhando a arquitetura da rede, o que limita nosso conhecimento sobre suas especificidades técnicas e impede uma compreensão mais aprofundada da arquitetura e das inovações da YOLOv8. Apesar disso, algumas informações estão disponíveis no repositório GitHub Ultralytics (2023b), como pode ser observado na Figura 2.16.

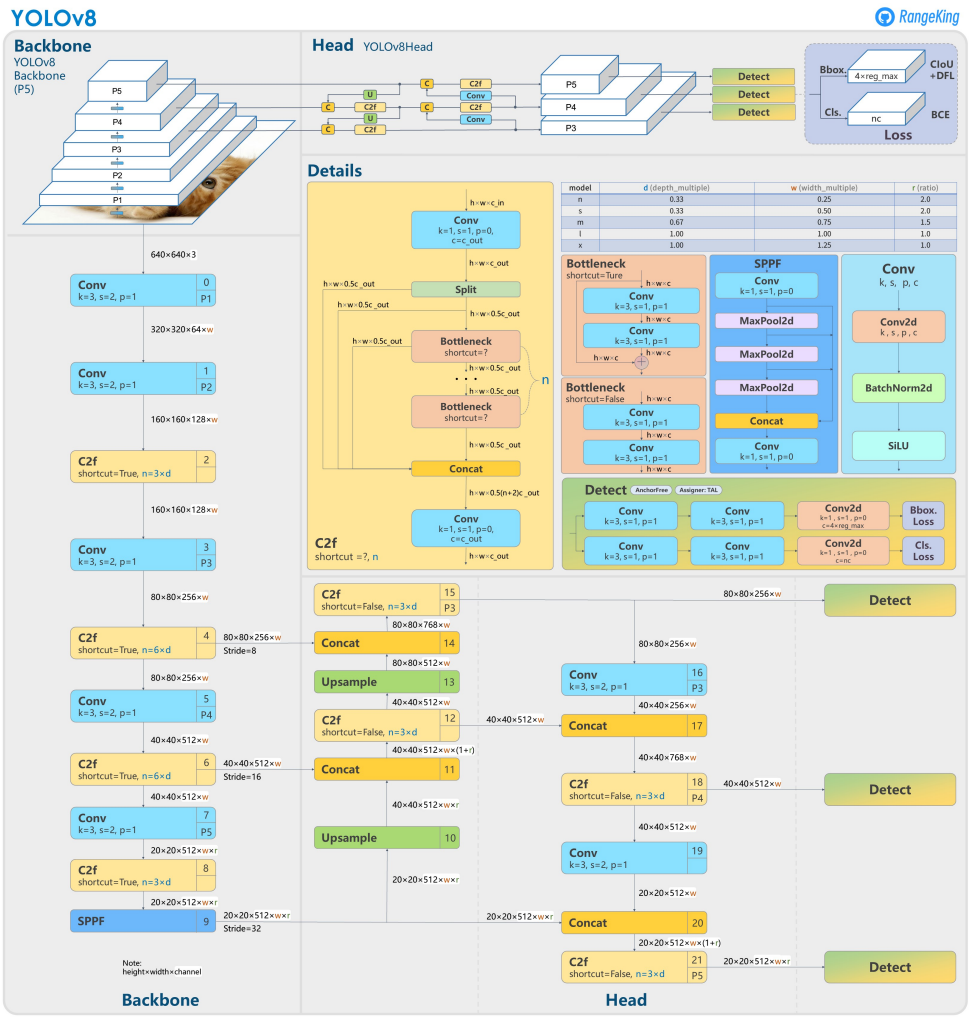


Figura 2.16 – Estrutura do modelo de detecção da YOLOv8

Fonte: (ULTRALYTICS, G., 2023a)

Tem-se, portanto, que as melhorias e inovações da YOLOv8, principalmente quando

comparada à versão YOLOv5 anterior da Ultralytics, incluem:

- ***Backbone*:** ainda há o uso do CSP, porém o módulo C3 da YOLOv5 é substituído pelo módulo C2f para uma maior otimização;
- ***Neck*:** usa-se o SPPF (*Fast SPP*), que é mais rápido que o SPP porque não repete as mesmas operações de *maxpool*, ele reutiliza operações anteriores para produzir uma saída idêntica ao SPP mas com uma implementação matematicamente mais eficiente;
- ***Head*:** possui um *design* desacoplado, isto é, ela processa as diferentes tarefas de detecção, regressão e classificação de maneira separada;
- ***Anchor-free*:** assim como a YOLOv6, a YOLOv8 traz a abordagem livre de âncoras, detectando diretamente o centro do objeto e não o centro em relação a uma *anchor box*, o que melhora a generalização para novos dados e aumenta a velocidade da rede;
- ***Mosaic Data Augmentation*:** aplica durante o treinamento mosaicos que consistem em unir quatro imagens, forçando o modelo a aprender objetos em novos locais, em oclusão parcial e com diferentes pixels ao redor.

Após apresentadas todas as versões da YOLO, a Tabela 2.1 resume as principais diferenças entre elas e a Figura 2.17 fornece uma linha do tempo ilustrando os anos criação de cada versão. Essas informações ajudarão a visualizar a evolução da YOLO ao longo do tempo e a compreender as diferenças implementadas em cada etapa do desenvolvimento dessa arquitetura de detecção de objetos.

Versão	***Backbone***	***Neck***	***Head***	***Loss Function***
YOLOv3	Darknet53	FPN	Âncoras	BCE
YOLOv4	CSPDarknet53	SPP e PANet	Âncoras	BCE/CIoU
YOLOv5	CSPDarknet53	PANet	Âncoras	BCE/CIoU
YOLOv7	EELAN	PANet	Líder/Auxiliar	BCE
YOLOv6	RepVGG/CSPRepStack	RepPAN	Desacoplada	VFL/DFL
YOLOv8	CSPDarknet53	SPPF	Desacoplada	BCE/DFL/CioU

Tabela 2.1 – Tabela comparativa das versões da YOLO

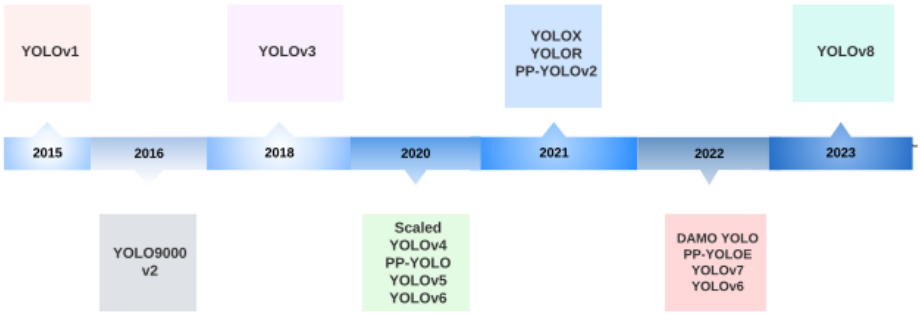


Figura 2.17 – Linha do tempo das versões da YOLO

Fonte: Terven e Cordova-Esparza (2023, p. 2)

Vale ressaltar que testar diferentes versões da YOLO em um conjunto de dados personalizado é interessante para maximizar a eficácia do sistema de detecção de objetos em tempo real. Cada versão da YOLO apresenta suas próprias inovações e ajustes, o que pode resultar em diferenças significativas no desempenho, dependendo das características específicas do conjunto de dados e das necessidades computacionais. A diversidade de objetos e imagens do conjunto de dados pode influenciar na eficácia do modelo, como por exemplo algumas podem ser melhor na detecção de objetos pequenos, enquanto outras podem ser mais adequadas para objetos maiores ou para imagens com alta densidade de objetos. Da mesma forma, algumas versões da YOLO podem lidar melhor com imagens de baixa resolução, enquanto outras podem ser otimizadas para imagens de alta resolução. Os requisitos computacionais também são bem diferentes a depender do modelo, sendo algumas mais adequadas para sistemas com recursos limitados, enquanto outras versões podem ser mais poderosas, mas exigem mais capacidade de processamento. Além disso, em termos de precisão e velocidade, algumas versões da YOLO podem priorizar a precisão, mesmo à custa da velocidade, enquanto outras podem ser otimizadas para fornecer detecções rápidas, mesmo que isso possa reduzir a precisão.

2.3 *Synthetic Data*

Uma abordagem comum para tentar aumentar a quantidade de dados de treinamento é a geração de dados sintéticos, a qual requer uma breve explicação. Com a função de mimetizar dados do mundo real, os dados sintéticos são gerados através das propriedades estatísticas dos dados originais (HEYBURN et al., 2018). Isso pode ser especialmente útil para sistemas que dependem de fontes de dados limitadas. Uma forma de gerar dados sintéticos é por meio de uma arquitetura de aprendizado de máquina não-supervisionado conhecida como *Generative Adversarial Network* (GAN). A arquitetura de uma GAN consiste na integração de 2 CNNs concorrentes: o gerador e o discriminador. Como o próprio nome

sugere, o gerador é responsável pela criação de novos dados que sejam realistas o suficiente a ponto de conseguir enganar o discriminador, cujo objetivo é melhorar sua capacidade de discernimento e diferenciar um dado *fake* de um dado proveniente do mundo real. Assim sendo, as duas redes competem e formam uma arquitetura de duplo *feedback*. A melhoria de uma rede implica em uma maior *loss* na outra, fazendo com que as duas melhorem concomitantemente. A Figura 2.18 ilustra a arquitetura padrão de uma GAN.

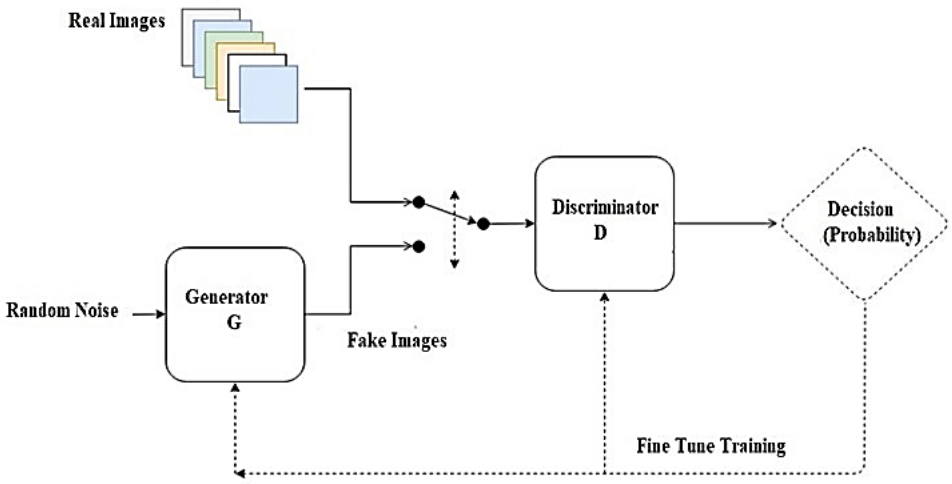


Figura 2.18 – Arquitetura padrão de uma GAN

Fonte: Research Gate

Metodologicamente, o gerador cria uma imagem de saída a partir da entrada de ruído. Essa imagem é então passada ao discriminador, o qual tendo acesso ao banco de dados de imagens reais, dá como saída a probabilidade da imagem ser verdadeira ou não, criando um erro que é posteriormente utilizado no ajuste de pesos.

As GANs tem sido muito utilizadas na geração de vídeos, imagens e músicas realistas que são muitas das vezes indistinguíveis de criações artísticas humanas. Existem diversas arquiteturas diferentes com propósitos cada vez mais específicos, como *Deep Convolutional Gan* (DCGAN), *Super Resolution Gan* (SRGAN), *CycleGan* e *Pix2Pix*. Entretanto, apesar da grande variabilidade de arquiteturas e das múltiplas possibilidades de aplicação, como criação artística e *Data Augmentation*, as GANs ainda possuem alguns desafios a serem superados, como um conjunto limitado de saídas possíveis e a dificuldade de treinamento, uma vez que requer grande quantidade de dados para resultados mais complexos.

2.4 *Transfer Learning*

No contexto das CNNs, *transfer learning* (aprendizado por transferência, em português) corresponde a utilizar o conhecimento obtido em um conjunto de dados para melhorar

o desempenho em outro conjunto de dados diferente, mas relacionado (Figura 2.19). A ideia por trás desta técnica é explorar as características e padrões aprendidos por um modelo pré-treinado, em vez de treinar um novo modelo do zero. Isso não só economiza recursos computacionais e tempo, mas também ajuda a obter um melhor desempenho. Isso é particularmente útil quando o conjunto de dados a ser utilizado possui dados rotulados limitados, já que o treinamento de um modelo do zero com poucos dados pode resultar em *overfitting* (ajuste excessivo) nos dados de treinamento. (HO; KIM, 2021).

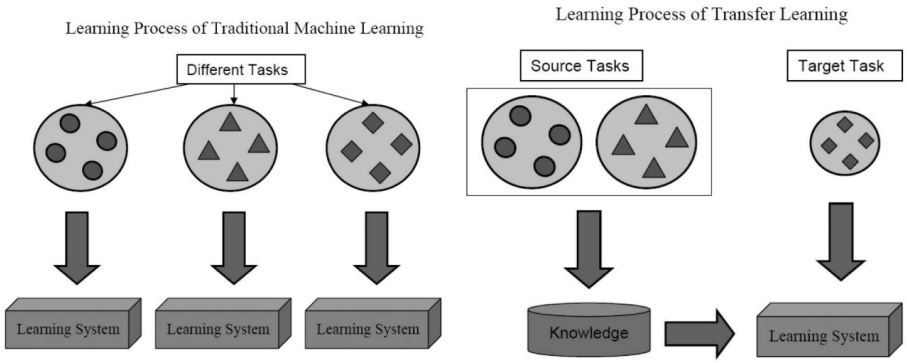


Figura 2.19 – Comparação entre o processo tradicional de aprendizado e o processo de aprendizado com o uso de *transfer learning* para diferentes tarefas

Fonte: Pan e Yang (2010, p. 1346)

O *transfer learning* é frequentemente implementado usando pesos de um modelo geralmente treinado em grandes conjuntos de dados como os *datasets* ImageNet (DENG et al., 2009) e COCO (LIN, T. et al., 2014), que contêm milhões de imagens rotuladas de várias classes. Durante o pré-treinamento, o modelo aprende a identificar uma série de características que são estruturadas de forma hierárquica. Isso significa que as características identificadas vão desde as mais elementares, como bordas e texturas (conhecidas como características de baixo nível), até as mais complexas e abstratas, como a identificação de partes específicas e classes de objetos (conhecidas como características de alto nível). Essas características aprendidas podem então ser reutilizadas para um novo conjunto de dados, pois fornecem um forte conhecimento prévio para o reconhecimento de objetos e outras tarefas relacionadas.

3 Trabalhos Relacionados

Há uma variedade considerável de estudos publicados sobre a utilização de RPAS na aquisição de imagens aéreas em diferentes culturas agrícolas. Essas pesquisas abrangem desde o imageamento aéreo puro, passando pela aplicação de técnicas clássicas de processamento de imagens, até a exploração de abordagens mais avançadas e inovadoras. Neste contexto, o presente trabalho propõe uma revisão aprofundada da literatura, com ênfase nos estudos que discutem o uso de AI para extrair informações valiosas e complexas das imagens aéreas obtidas com o auxílio de RPAS. A análise desses estudos permitirá compreender as potencialidades e limitações das técnicas de AI no tratamento de dados coletados por RPAS.

3.1 Uso de RPAS para Imageamento Aéreo

No estudo realizado por Cardoso, Farias e João Almiro Corrêa Soares (2022), é apresentada uma análise abrangente do emprego de UAVs no cultivo de cana-de-açúcar no Brasil. O artigo detalha diversas aplicações dos UAVs, como a obtenção de dados de alta resolução, mapeamento da variabilidade espacial, identificação de ervas daninhas e avaliação do estresse das plantas. Os autores também exploram a sinergia entre os UAVs e outras tecnologias de agricultura de precisão, como sensores de solo e imagens de satélite. O trabalho aborda os desafios e oportunidades associados ao uso de UAVs no cultivo de cana-de-açúcar, enfatizando a importância da padronização dos protocolos de voo e processamento de dados para garantir resultados consistentes e precisos.

A pesquisa destaca a necessidade de estabelecer uma sólida base de conhecimento técnico e científico para maximizar o potencial dos UAVs nesse contexto. De modo geral, o artigo oferece uma visão completa do emprego de UAVs no cultivo de cana-de-açúcar no Brasil, abordando as principais aplicações, desafios e oportunidades para essa inovadora tecnologia agrícola. A Figura 3.20, extraída do artigo, ilustra a aplicação do imageamento aéreo realizado pelos UAVs em diferentes regiões do Brasil, demonstrando a versatilidade dessa tecnologia emergente.

Figura 3.20 – Aplicações e Trabalhos realizados com UAV em plantações de cana-de-açucar no Brasil

Fonte: Cardoso, Farias e João Almiro Corrêa Soares (2022, p. 1639)

3.2 Técnicas Clássicas na Análise de Imagens Aéreas

No estudo de Ji e Qi (2011) , os autores apresentam um algoritmo para detecção de linhas de cultivo baseado na transformada de Hough aleatória. A metodologia proposta emprega uma abordagem de amostragem aleatória para identificar linhas de cultivo em imagens capturadas manualmente com uma câmera. O artigo detalha a implementação do algoritmo e exibe resultados experimentais que demonstram a eficácia do método na detecção de linhas de cultivo. Entretanto, a aplicação do algoritmo é limitada a imagens obtidas ao nível do solo através de fotografias manuais.

Por outro lado, os autores Guilherme Afonso Soares, Abdala e Escarpinati (2018) propõem uma abordagem para identificar linhas de plantio em imagens aéreas adquiridas por meio de um UAV. O método proposto pelos autores segmenta as imagens em pequenos mosaicos, permitindo uma análise minuciosa das características das linhas de plantio. Posteriormente, a Transformada de Hough é aplicada a cada mosaico para identificar as linhas de plantio. Essa abordagem apresentou alta precisão na identificação das linhas e uma boa capacidade de lidar com imagens de baixa resolução e ruído. Os resultados obtidos sugerem

que a metodologia constitui uma solução viável e precisa para a identificação de linhas de plantio em imagens aéreas.

3.3 Técnicas de Inteligência Artificial (*Deep Learning*) na Análise de Imagens Aéreas

O artigo de Bah, Hafiane e Canals (2018) já traz mais informações sobre as técnicas que serão utilizadas neste trabalho. Ele propõe uma abordagem para detectar ervas daninhas em imagens aéreas de culturas agrícolas usando técnicas de *Deep Learning*. O método é baseado em uma abordagem de treinamento não supervisionado para rotular os dados de imagem e um modelo de CNN para realizar a detecção de ervas daninhas. A abordagem não supervisionada para rotulação de dados é usada para superar a necessidade de anotações manuais de imagem, que são trabalhosas e caras. Ela começa com a segmentação dos pixels da imagem em regiões de interesse usando um método de segmentação baseado em bordas, a qual é usada para identificar regiões de interesse que contêm objetos distintos na imagem, como plantas e ervas daninhas.

Em seguida, os pixels dentro das regiões de interesse são agrupados usando um algoritmo de agrupamento não supervisionado, que divide os pixels em diferentes classes com base em suas características. Essas classes podem ser vistas como diferentes tipos de objetos na imagem, como plantas, ervas daninhas e solo. Os *clusters* de pixels são usados para rotular as imagens de treinamento para o modelo de detecção de ervas daninhas. Este modelo é um modelo de CNN treinado com as imagens rotuladas, o qual é projetado para identificar regiões da imagem que contêm ervas daninhas com base nas características aprendidas durante o treinamento e é capaz de detectar ervas daninhas em imagens de diferentes tipos de solo e em diferentes condições climáticas.

Os resultados mostram que a abordagem proposta é capaz de detectar ervas daninhas com uma precisão em torno de 90% em um conjunto de dados de imagens de culturas agrícolas. Isso demonstra que o método pode ser uma solução eficaz para a detecção automatizada de ervas daninhas em imagens de culturas agrícolas, o que pode ajudar a melhorar a eficiência da agricultura e reduzir a necessidade de uso de herbicidas.

Um outro artigo dos mesmos autores publicado dois anos depois deste citado anteriormente, Bah, Hafiane e Canals (2020), descreve um método para detecção automática de linhas de plantio em imagens capturadas por UAVs, com o objetivo de ajudar na agricultura de precisão. O método proposto, CRowNet, é composto de duas etapas principais: a segmentação de imagem baseada na rede neural SegNet e a detecção de linhas de plantio baseada na combinação da CNN com a transformada de Hough. A segmentação de imagem é usada para separar as plantas do resto da imagem, melhorando a detecção de linhas de plantio. Para isso, a SegNet é usada para gerar uma máscara binária das plantas na imagem. Em seguida, a

detecção de linhas de plantio é realizada usando a combinação da CNN com a transformada de Hough. A transformada de Hough é usada para identificar múltiplas linhas nas imagens, e depois a CNN é responsável por detectar as principais linhas que correspondem as linhas do plantio de fato. O uso da HoughCNet, que integra a transformada de Hough à CNN, permite que as duas abordagens sejam combinadas de forma mais eficiente.

Os resultados experimentais mostram que o método proposto superou outros métodos de detecção de linhas de plantio em termos de precisão. O CRowNet atingiu uma taxa de detecção de 93,58% , o que o torna adequado para aplicações em larga escala em agricultura de precisão. Os autores também compararam o desempenho do CRowNet com outras abordagens de detecção de linhas de plantio, como aquelas baseadas na transformada de Hough e na segmentação de imagem, e demonstraram que o CRowNet apresentou melhores resultados em termos de precisão. Tem-se então que o CRowNet é um método promissor para a detecção automática de linhas de plantio em imagens aéreas de culturas, que pode ser usado para melhorar a eficiência e precisão da agricultura de precisão.

O trabalho em Silva, Cielniak e Gao (2021) apresenta um método para detecção de linhas de plantações agrícolas em imagens obtidas em condições de campo variáveis utilizando *deep learning*. O sistema proposto usa CNN para detectar as linhas de culturas em imagens de plantações adquiridas por uma câmera RGB montada em um veículo agrícola autônomo. Para treinar o modelo, os autores usaram um conjunto de imagens rotuladas, que incluíam informações sobre a localização das linhas de culturas. O modelo foi então ajustado para aprender a detectar essas linhas de culturas nas imagens. Para melhorar a capacidade do modelo de lidar com variações nas condições do campo, os autores também usaram técnicas de *data augmentation*, como rotação, espelhamento e mudança de brilho, para gerar mais exemplos de treinamento. Isso ajudou a tornar o modelo mais robusto em relação a variações na iluminação, condições climáticas e tipos de plantações.

A rede U-Net foi adaptada para realizar a detecção de linhas de culturas em imagens agrícolas. A rede *encoder* foi treinada para extrair características visuais relevantes para a detecção de linhas de culturas e a rede *decoder* foi treinada para gerar uma máscara de segmentação que destaca as linhas de culturas. Os autores utilizaram os parâmetros base da rede U-Net e demonstraram que ela é capaz de detectar com precisão as linhas de culturas em uma ampla variedade de condições de campo. A avaliação do método em diferentes condições de campo teve como resultado uma taxa de detecção média de 89,36%. Esse trabalho é um passo importante em direção à automação de operações agrícolas, o que pode levar a uma maior eficiência e redução de custos, além de permitir uma agricultura mais precisa e sustentável.

4 Metodologia e Desenvolvimento

Nesta seção, apresentaremos a metodologia adotada para a realização deste trabalho. Para isso, foi concebido o diagrama apresentado na Figura 4.21, com o objetivo de proporcionar uma visão clara e abrangente de cada etapa realizada, além de destacar a sequência em que as etapas do trabalho foram desenvolvidas.

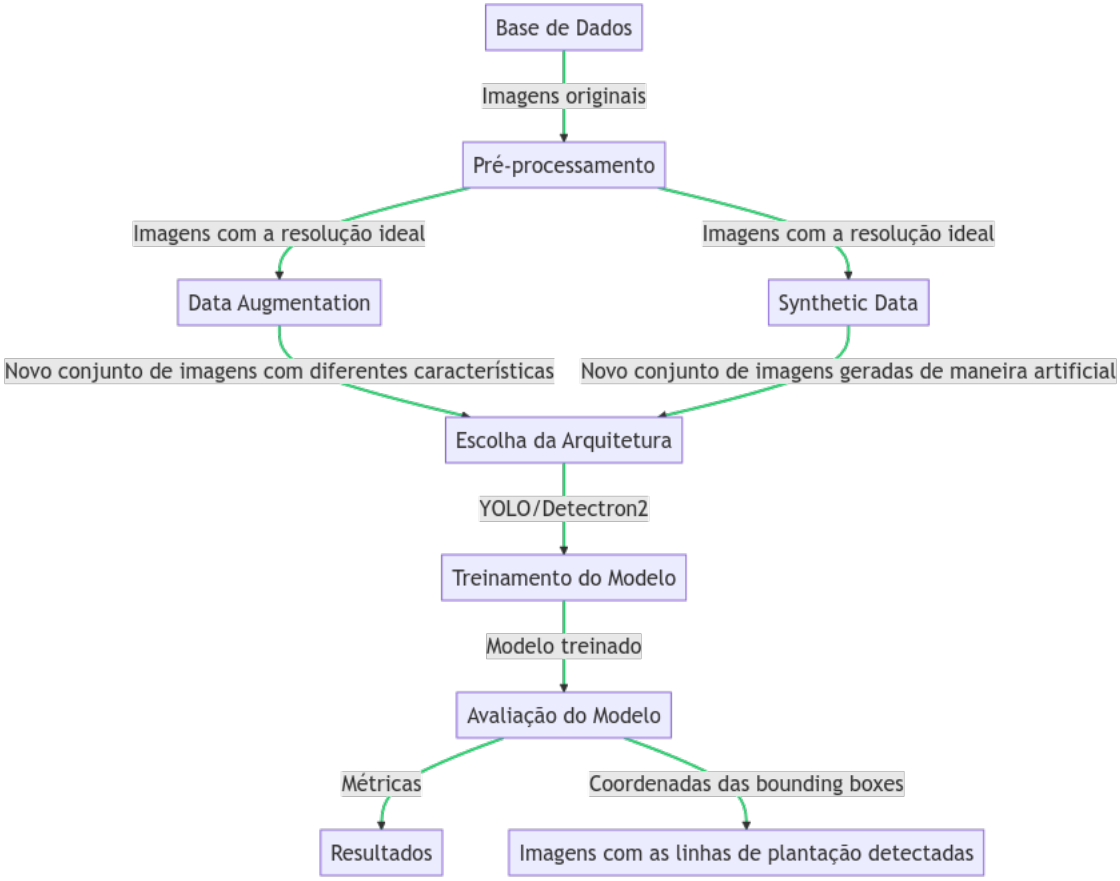


Figura 4.21 – Etapas para a realização deste trabalho

Fonte: autores

A seguir, faremos uma análise minuciosa de cada componente do diagrama, explicando o propósito de cada etapa e como foram executadas.

4.1 Base de Dados

Inicialmente, a coleta das imagens utilizadas nesse trabalho foi realizada através de uma câmera acoplada em um UAV comercial em lavouras de cana-de-açúcar em fase

de colheita. Ao todo, tínhamos acesso a apenas 40 imagens. Um exemplo de imagem se encontra na Figura 4.22.

Figura 4.22 – Exemplar de imagem capturada pelo UAV

Fonte: Imagem adquirida através do UAV

Conforme prosseguíamos com o trabalho, percebemos que a baixa quantidade de imagens poderia enviesar a performance da CNN utilizada, mesmo com a aplicação de *Data Augumentation*. Outro motivo que nos levou a tomar essa decisão é a existência de imagens com carência de características interessantes para observação, o que consequentemente dificultaria uma posterior rotulagem. Essa falta de informações coletáveis pode ser exemplificada na Figura 4.23, da qual não conseguimos anotar *features* como linha, ângulo, falha, dentre outros.

Figura 4.23 – Exemplar de imagem precária

Fonte: Imagem adquirida através do UAV

Dessa forma, optamos pela busca de um *dataset* que contivesse imagens semelhantes, possibilitando maior aproveitamento do contexto da pesquisa e dos estudos já realizados, o que nos levou a escolha das imagens presentes no trabalho de Ribera et al. (2019).

Esse *dataset* consiste em 60 mil imagens de plantação de sorgo capturadas por um UAV sobrevoando a uma altura de 40 metros, totalizando 15.208 plantas de sorgo. Todas as imagens já estão rotuladas, associando a cada imagem uma lista contendo as coordenadas em pixeis de cada planta na foto. As imagens obtidas foram costuradas de maneira a formar um ortomosaico de dimensão 6.000x12.000 com uma resolução de 0.75cm/pixel. As 60 mil imagens foram segregadas em 3 grupos: 50.000 imagens de treino, 5.000 de validação e 5.000 de teste. O mesmo *dataset* também foi disponibilizado com imagens de resolução quadrada 256x256, ideal para o treinamento de CNNs.

A implementação de um modelo capaz de detectar pontos se mostrou bastante complexa conforme o prosseguimento do trabalho, visto que não existem muitas arquiteturas bem documentadas com esse objetivo. Então, decidimos optar pela detecção das linhas das plantações.

Além disso, uma quantidade muito grande de imagens de um *dataset* no qual não se tem informações detalhadas sobre todas as imagens presentes nele, dificulta o controle da qualidade das imagens de treino, gerando futuramente um desempenho insatisfatório do modelo que poderá aprender padrões errados e inconsistentes. Dessa maneira, optamos por selecionar manualmente 298 imagens do *dataset* escolhido. Utilizando a plataforma de rotulagem *LabelBox* (LABELBOX, s.d.), rotulamos as linhas das plantações com caixas delimitadoras, ou *bounding boxes*. Em seguida, utilizando a plataforma de visão computacional *Roboflow* (ROBOFLOW, s.d.), segregamos a base de dados em 241 imagens de treino, 25 imagens de validação e 32 imagens de teste, seguindo a proporção de aproximadamente 80% : 10% : 10%, respectivamente.

4.1.1 Pré-processamento (*resizing*)

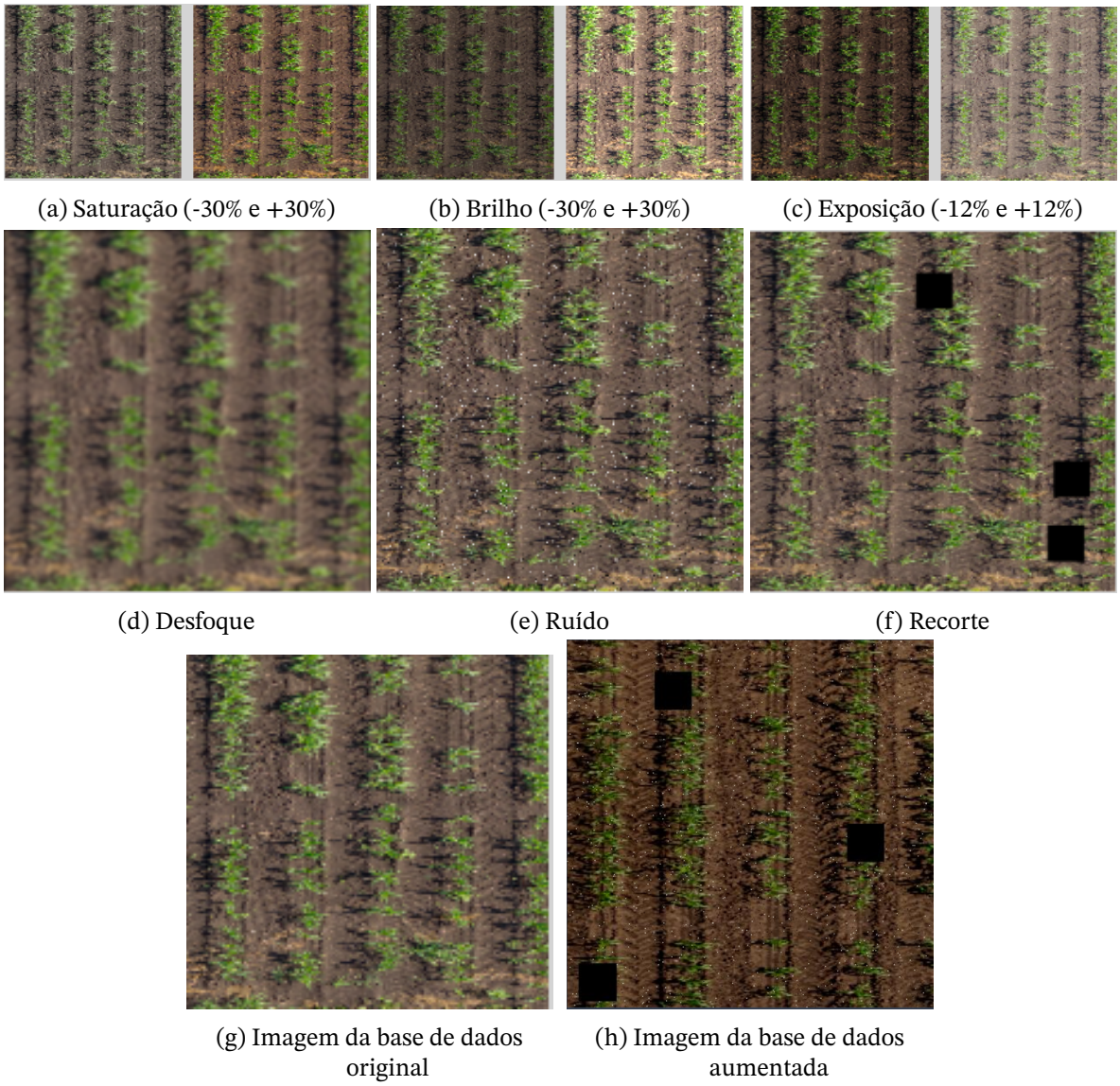
A resolução ideal da imagem para o treinamento de uma CNN é influenciada por vários fatores, tais como a visibilidade das características desejadas, a arquitetura da rede e a capacidade de processamento computacional disponível. Idealmente, uma boa resolução preserva informações importantes das características, mas também é compacta o suficiente para permitir um treinamento eficiente. No entanto, é importante observar que a resolução ideal pode variar dependendo do contexto. Para o nosso trabalho, inicialmente testamos uma resolução de 640x640, a qual gerou um sobrecarregamento da RAM e um treino muito delongado. Assim sendo, escolhemos uma resolução de exportação menor de 416x416, cujo tamanho é ideal para o treinamento das primeiras arquiteturas de CNNs que testamos posteriormente.

4.1.2 *Data augmentation*

A quantidade de imagens necessárias para atingir um determinado desempenho depende tanto da complexidade da tarefa quanto da eficiência da arquitetura da CNN implementada. Para aumentar a quantidade de imagens utilizadas para o treino do nosso modelo, foi utilizada a técnica de *Data Augumentation*, que consiste em selecionar uma imagem e, através de combinações randômicas de transformações simples gerar imagens derivadas (SHORTEN; KHOSHGOFTAAR, 2019). Como decisão de projeto, foram geradas 3 imagens a partir de cada imagem de treinamento original, resultando num banco de dados contendo 780 imagens, sendo 723 de treino, 25 de validação e 32 de teste. O processo foi realizado por meio do próprio *Roboflow*(ROBOFLOW, s.d.).

A utilização de diferentes técnicas de *Data Augmentation* torna o modelo mais robusto às variações no ambiente, especialmente considerando que as imagens são fotografias aéreas agrícolas capturadas por um drone. Ao aplicar diversas transformações nas imagens, o modelo se adapta melhor a mudanças que podem ocorrer no ambiente real. As técnicas de *Data Augmentation* utilizadas neste projeto podem ser visualizadas na Figura 4.24 e incluem:

- **Saturação (variação de -30% a +30%):** ajustar a saturação das cores das imagens ajuda o modelo a lidar com variações na qualidade da luz e na aparência das cores das plantações em diferentes condições climáticas e horários do dia;
- **Brilho (variação de -30% a +30%):** alterar o brilho das imagens permite que o modelo se adapte a mudanças na iluminação, como um dia nublado ou ensolarado, e às variações na intensidade da luz solar ao longo do dia;
- **Exposição (variação de -12% a +12%):** modificar a exposição das imagens é útil para treinar o modelo a lidar com condições de luz variáveis e possíveis erros de exposição nas imagens capturadas pelo drone;
- **Desfoque (até 1 pixel):** adicionar desfoque às imagens ajuda o modelo a ser mais tolerante a imagens com baixa qualidade de foco, seja devido a problemas no equipamento, movimento do drone ou condições atmosféricas adversas;
- **Ruído (até 2% dos pixels):** introduzir ruído nas imagens simula imperfeições na captura de imagens pelo drone, como interferências eletromagnéticas, e ajuda o modelo a ser mais resistente a pequenas perturbações na qualidade das imagens;
- **Recorte (3 caixas com 10% de tamanho cada):** aplicar recortes aleatórios nas imagens força o modelo a aprender a identificar características relevantes em diferentes partes da imagem, tornando-o mais eficiente na detecção de plantações mesmo com possíveis oclusões parciais causadas por nuvens, construções ou outros elementos do ambiente.



(a) Saturação (-30% e +30%) (b) Brilho (-30% e +30%) (c) Exposição (-12% e +12%)

(d) Desfoque (e) Ruído (f) Recorte

(g) Imagem da base de dados original (h) Imagem da base de dados aumentada

Figura 4.24 – Imagem da base de dados aumentada e exemplos da aplicação das transformações de *Data Augmentation* em uma imagem da base de dados original.

Fonte: autores

Ao incorporar essas técnicas de *Data Augmentation* no treinamento do modelo, espera-se obter uma performance melhor e mais consistente do modelo em diferentes condições. Consequentemente, espera-se obter melhorias na robustez e na capacidade de generalização em relação aos desafios encontrados no ambiente real das imagens aéreas agrícolas.

4.1.3 Geração de dados sintéticos via GAN

A priori, a motivação de gerar dados sintéticos via GAN é mimetizar os dados reais e incluir os resultados no conjunto de dados de treinamento. Assim, é possível que o conjunto resultante seja mais diversificado, fazendo com que o modelo final seja mais generalista e performe melhor quando submetido à dados que ainda não foram apresentados. Nesse

viés, um arquitetura de DCGAN foi implementada via adaptação de um *script* em Python fornecido por Shakhadri (s.d.). Para o treinamento da GAN, foi utilizado o mesmo *dataset* já introduzido nesse documento.

Com o fito de explorar melhor os resultados gerados pela GAN, testamos três resoluções de saída: 64×64, 128×128 e 256×256. Como será discutido posteriormente, os resultados gerados pela resolução de 256×256 ficaram ininteligíveis, enquanto que a resolução de resolução 64×64 destoa muito da resolução utilizada de fato pela rede. Nesse viés, criou-se uma cópia do conjunto de dados de treino original, porém agora com a adição de 28 imagens sintéticas de resolução 128×128. Esse novo conjunto de dados alimentará somente o modelo final com o fito de verificar se uma base de dados com algumas imagens sintéticas pode melhorar o aprendizado e a capacidade de generalização do modelo. A nível de modelo, a rede executa 1000 épocas com um *batch size* de 32 imagens retiradas do *dataset* de treinamento isento de *Data Augmentation*. Outrossim, é importante frisar que a cada época, a rede gera 28 imagens.

4.2 Apuração de Modelos e Otimização

O Detectron2 (WU et al., 2019b) é um *framework* de código aberto do FAIR que fornece diversos algoritmos para detecção e segmentação de objetos. O Detectron2, amplamente utilizado pela comunidade de pesquisa, surge após seu antecessor, o Detectron, fornecendo mais velocidade, flexibilidade e facilidade de uso. A plataforma é construída em PyTorch, biblioteca muito utilizada de *deep learning* em Python, fornecendo diversas arquiteturas de CNNs relevantes para o nosso trabalho. Nesse sentido, escolhemos explorar os dois modelos já previamente apresentados na seções 2.2.1 e 2.2.2: *Faster R-CNN* e *RetinaNet*, respectivamente. Outrossim, as arquiteturas YOLO apresentadas na seção 2.2.3, serão utilizadas via seus respectivos *frameworks*.

Vale ressaltar que o uso de diferentes arquiteturas e *frameworks* permite explorar as vantagens e desvantagens de cada implementação com o fito de selecionar a melhor arquitetura para dar continuidade ao projeto, tendo em vista um mesmo ambiente de testes. Nesse viés, otimizaremos a referida arquitetura com o fito de obter desempenho superior, levando em consideração as métricas avaliativas que serão descritas na sequência.

4.2.1 Métricas de Avaliação

Na avaliação de modelos de CNNs para detecção de objetos, é importante entender os conceitos de Verdadeiro Positivo (TP, na sigla em inglês), Falso Positivo (FP, na sigla em inglês) e Falso Negativo (FN, na sigla em inglês). Estes são termos usados para descrever os resultados de uma previsão do modelo em relação aos valores reais (verdadeiros).

- **TP:** um verdadeiro positivo ocorre quando o objeto foi corretamente identificado pelo modelo;
- **FP:** um falso positivo ocorre quando o modelo identifica um objeto onde não havia nenhum;
- **FN:** um falso negativo ocorre quando o modelo não identifica um objeto que estava presente.

Um outro conceito é o de Verdadeiro Negativo (TN, na sigla em inglês). No entanto, neste trabalho, que envolve apenas uma classe, isso não é relevante porque estamos interessados apenas na presença ou ausência de um tipo de objeto.

A seguir, discutiremos algumas métricas importantes usadas para avaliar e melhorar modelos de detecção de objetos (ELAIDOUNI, 2021):

- ***Precision*:** é uma medida de quantas detecções positivas do modelo estão realmente corretas. Ela é definida pela equação 4.1;

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

- ***Recall*:** também chamada de sensibilidade, é uma medida de quantas das detecções verdadeiras e positivas o modelo conseguiu identificar corretamente. O cálculo desta métrica se da por meio da equação 4.2;

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

- ***F1-score*:** é a média harmônica entre precisão e *recall*. É uma métrica útil quando se busca equilibrar *precision* e *recall*, como apresentado na equação 4.3;

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{4.3}$$

- ***Intersect over Union* (IoU):** é uma medida de *overlap* (sobreposição) entre duas *bounding boxes*, como pode ser observado pela equação 4.4;

$$IoU = \frac{Area\,of\,Overlap}{Area\,of\,Union} \tag{4.4}$$

- ***Average Precision* (AP):** é uma métrica que calcula a área sob a curva de *precision-recall*;
- ***Mean Average Precision* (mAP):** esta métrica calcula a precisão média de todas as classes de objetos;

- **Loss:** A função de perda é uma forma de medir o quão longe as previsões do modelo estão dos resultados reais. O processo de treinamento visa reduzir esta métrica;
- **Índice de confiança**: valor pertencente ao intervalo entre 0 e 1 referente à probabilidade de um objeto estar contido em uma caixa delimitadora.

Vale ressaltar que, em situações onde estamos lidando com apenas uma classe de objeto, como no caso presente, as métricas mAP e AP tornam-se equivalentes. Isso ocorre porque o mAP é a média das APs para todas as classes. Portanto, quando temos apenas uma classe, o mAP é apenas a AP dessa classe.

A Figura 4.25 traz uma melhor visualisação das métricas de *Precision*, *Recall* e IoU.

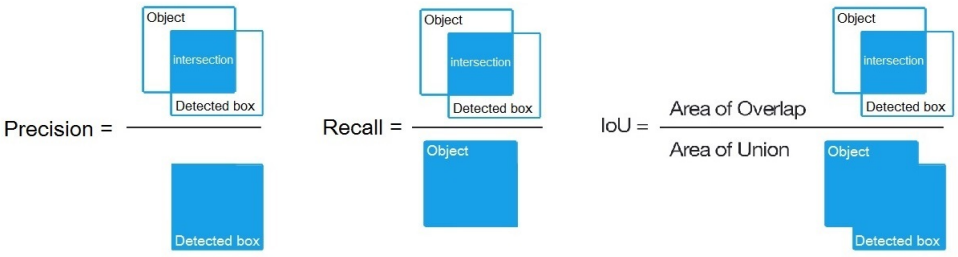


Figura 4.25 – *Precision*, *Recall* e IoU

Fonte: (ALEXEYAB, 2021)

4.2.2 Treinamento dos Modelos

Inicialmente, para o treinamento dos modelos utilizados nesse estudo, é feita uma seleção dos hiperparâmetros que serão utilizados, os quais irão impactar na capacidade do modelo de aprender efetivamente a partir dos dados e, consequentemente, influenciar na eficácia do modelo em predições. Os hiperparâmetros base fixados para a primeira rotina de testes são como segue:

- **Número de épocas:** se refere ao número de vezes que todos os dados presentes sejam processados pela rede. Cada fim de época implica em ajuste de pesos via comparação entre a saída desejada e a saída esperada. É necessário buscar, empiricamente, valores que melhorem a acurácia do modelo. A priori, épocas excessivas podem tomar muito tempo para treinar e, possivelmente, gerar *overfitting*. A posteriori, escassez de épocas pode limitar o aprendizado da rede neural, além de gerar um possível *underfitting* devido à dificuldade em capturar padrões complexos nos dados;
- ***Batch size* (tamanho do lote, em português):** o *batch size* determina quantas amostrar presentes na base de treinamento serão processadas a cada iteração. Tamanhos de lote muito grande exigem mais memória, tempo e são mais propensos a *overfitting*.

Por outro lado, lotes muito pequenos exigem menos memória e tempo, mas podem também atrapalhar no aprendizado do modelo;

- ***Learning rate* (taxa de aprendizado, em português):** o *learning rate*, ou LR, influencia diretamente na magnitude da atualização dos pesos ao fim de cada época. Uma taxa de aprendizado muito alta pode implicar num baixo desempenho, visto que impede que o modelo alcance pontos de mínimo na função de perda. Uma taxa de aprendizado muito baixa requisita mais épocas para que haja tempo para o modelo aprender. Além disso, o modelo pode ter dificuldades em "sair"de um ponto de mínimo local da função de perda;

- **Otimizador:** são algoritmos cuja função é atualizar os pesos de modo a minimizar a função de perda durante o processo de treinamento. Os otimizadores utilizados nesse projeto são o ADAM e o SGD.

Além disso, neste estudo, foi utilizado o *transfer learning* (discutido na seção 2.4), de forma a adaptar os pesos pré-treinados para o nosso *dataset* em específico, economizando tempo e recursos computacionais. É importante ressaltar que nem todas as configurações são iguais. Isso se deve à necessidade de adaptar certos hiperparâmetros em função das restrições de recursos computacionais. Em alguns casos, a complexidade dos modelos exigiu a alteração dos hiperparâmetros para permitir que o treinamento ocorresse dentro de uma escala de tempo viável ou para garantir que o treinamento fosse possível, dada a capacidade de processamento disponível.

Em primeira mão, analisaremos possíveis variações das arquiteturas *RetinaNet* e *Faster R-CNN* disponibilizadas pelo Detectron2. Para a *Faster R-CNN*, temos 3 tipos de *backbone* (FPN + ResNet, C4 e DC5), enquanto que para a *RetinaNet* temos apenas um (FPN+ResNet). Como citado por Wu et al. (2019a), a configuração FPN + ResNet apresenta melhor *trade-off* velocidade/acurácia, sendo escolhida tanto para *RetinaNet* quanto para a *Faster R-CNN*. Além disso, o Detectron2 disponibiliza a ResNet50 e a ResNet101 com 50 e 101 camadas, respectivamente. Por fim, também é possível escolher entre modelos muito treinados (3x) e modelos com pouco treino (1x), esse último não sendo relevante para nosso projeto. Assim, as variações de arquiteturas possíveis no Detectron2 e seus respectivos hiperparâmetros podem ser sumarizados na tabela 4.2.

Modelo	**Épocas**	**Batch Size**	**LR**	**Otimizador**	**Pré-Treino**
Faster R-CNN-R50	5000	4	0.001	SGD	COCO
Faster R-CNN-R101	5000	4	0.001	SGD	COCO
RetinaNet-R50	5000	4	0.001	SGD	COCO
RetinaNet-R101	5000	4	0.001	SGD	COCO

Tabela 4.2 – Tabela dos hiperparâmetros utilizados no treinamento dos modelos Detectron2

Pelo fato de todas as arquiteturas acima estarem na mesma plataforma, optou-se por padronizar todos os hiperparâmetros de modo a obter uma comparação justa entre os modelos. Outrossim, os modelos YOLO utilizados no projeto podem ser sumarizados na tabela 4.3.

Modelo	**Épocas**	**Batch Size**	**LR**	**Otimizador**	**Pré-Treino**
YOLOv3	5000	64	0.001	SGD	COCO
YOLOv4	5000	64	0.001	SGD	COCO
YOLOv5m	100	16	0.0001	SGD	COCO
YOLOv7	5000	64	0.0001	SGD	COCO
YOLOv6m	100	32	0.000384	SGD	COCO
YOLOv8m	100	16	0.0001	SGD	COCO

Tabela 4.3 – Tabela dos hiperparâmetros utilizados no treinamento dos modelos YOLO

Os modelos apresentados foram então testados com o *dataset* explicitado na seção 4.1

4.2.3 Seleção da Arquitetura YOLOv8

Como será posteriormente discutido na seção 5, a arquitetura com maior AP foi a YOLOv4. Entretanto, optamos por prosseguir o projeto utilizando a arquitetura da YOLOv8, disponibilizada pela Ultralytics, fato que se explica pelos três seguintes motivos:

1. A YOLOv8 obteve um valor de AP suficientemente próximo ao valor obtido pela YOLOv4, o que impactaria quase que de maneira insignificante no nosso sistema final;
2. Sobretudo por ser tratar de uma versão mais recente, a YOLOv8, possui melhor documentação, praticidade de uso e uma gama de métricas e gráficos facilmente coletáveis já disponibilizados pelo *framework*, o que enriquece a parte de resultados;
3. Por possuir diversos tamanhos e fácil otimização (*tuning*), o modelo tem potencial para superar o AP obtido pela YOLOv4.

Assim sendo, percebemos um *trade-off* vantajoso na seleção da YOLOv8 como a arquitetura que será de fato escolhida para o projeto.

4.2.4 Otimização do Modelo Selecionado

Como mencionado na subseção 2.2.3, a YOLOv8 possui diferentes tamanhos de sua arquitetura, sendo *Nano*, *Small*, *Medium*, *Large*, *Extra Large*. Cada profundidade afeta diretamente tanto a velocidade de inferência quanto a precisão da rede, como pode ser observado na Figura 4.26.

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figura 4.26 – Comparação entre os diferentes tamanhos da YOLOv8 no COCO *dataset*

Fonte: (ULTRALYTICS, G., 2023b)

Como pode ser observado na Tabela 4.3, foi testado o modelo da YOLOv8m, pois ele representa um equilíbrio entre a precisão e a velocidade da rede. Após isso, todos os outros tamanhos da YOLOv8 foram testados com as mesmas configurações de hiperparâmetros utilizadas para YOLOv8m.

A escolha do tamanho ótimo do modelo foi motivada pela necessidade de maximizar a precisão dos resultados, de forma que a YOLOv8x foi a escolhida por apresentar o maior AP, conforme será demonstrado na seção 5.3.1.

Outra maneira de aprimorar o modelo é através da variação dos hiperparâmetros introduzidos na subseção 4.2.2, o que sugere que a variação sistemática dos mesmos pode melhorar significantemente o desempenho do modelo resultante. Para isso, foi realizada uma busca em grade, na qual para cada hiperparâmetro é definido um conjunto de valores e todas as combinações são posteriormente avaliadas. Entretanto, o número de épocas não foi variado, sendo fixado em 30, tendo em vista a necessidade de otimizar o tempo diante da grande quantidade de combinações. A ideia inicial era obter os melhores hiperparâmetros e só então realizar o treinamento completo com 100 épocas. Todavia, o treinamento completo mostrou um desempenho inferior ao treinamento parcial, ocasionando na fixação de 30 épocas.

A busca em grade realizada pode ser sumarizada pelos valores fornecidos na Tabela 4.4.

Hiperparâmetros	**Valores**
Learning rate	0.0001, 0.001, 0.01
Batch size	8, 16, 32, -1
Otimizador	Adam, SGD

Tabela 4.4 – Tabela dos valores de hiperparâmetros utilizados na otimização

Note que um dos valores passados para o *batch size* é -1, significando que o mesmo será calculado conforme a disponibilidade de memória da GPU responsável pelo processamento. O cálculo é realizado através do arredondamento de 66,67% da memória da GPU, em GBs. Em síntese, essa etapa final de otimização consiste no treinamento e validação de 24 modelos resultantes das combinações dos hiperparâmetros. Como será visto na seção 5.3.2, o melhor conjunto de hiperparâmetros é apresentado na Tabela 4.5.

Hiperparâmetros	**Valores**
Learning rate	0.001
Batch size	16
Otimizador	SGD

Tabela 4.5 – Tabela dos melhores valores de hiperparâmetros

4.3 Algoritmos para Identificação de Falhas e Cálculo do Aproveitamento do Solo

Para uma melhor análise dos resultados obtidos, foram desenvolvidos dois algoritmos (Apêndice B) baseadas em dois diferentes formatos das coordenadas das *bounding boxes* fornecidos pela rede. Esses formatos são:

- $x_1y_1x_2y_2$: representam as coordenadas do canto superior esquerdo e do canto inferior direito da *bounding box*, respectivamente;
- $xywh$: representam as coordenadas do ponto central da *bounding box*, tanto quanto sua largura e altura.

Com essas duas formas de coordenadas, pode-se explorar dois importantes aspectos das imagens, as falhas nas linhas de plantação e a porcentagem da área coberta pelas plantas.

Tendo como referência uma determinada *bounding box*, são consideradas falhas: lacunas entre outras *bounding boxs* verticalmente alinhadas e/ou entre o limite superior/inferior da imagem. Essas falhas são então mensuradas e quantificadas, proporcionando uma avaliação precisa da disposição e densidade das plantas no campo plantado. Essa informação torna-se fundamental para identificar partes da lavoura que possam necessitar de replantio ou cuidados adicionais.

Quanto ao cálculo do aproveitamento do solo de cada imagem, é realizado o somatório das áreas de todas as *bounding boxes*, o qual é dividido pela área total de imagem, ambos em pixels. Esse procedimento representa a porcentagem de ocupação das k plantas na imagem, e é formalizado na equação 4.5.

$$PA_n = \frac{\sum_{i=1}^{i=k} w_k \cdot h_k}{AreaTotal} \tag{4.5}$$

Onde PA_n é a representação do percentual de aproveitamento do solo da enésima imagem, w_k e h_k são a largura e a altura da k-ésima *bounding box*, respectivamente, e $AreaTotal$ se dá pela quantidade total de pixels na imagem ($416 \times 416 = 173056$). Em seguida, para obtermos uma noção mais geral da lavoura como um todo, é calculado $\overline{PA}$ como sendo a média dos PAs de todas as n imagens (vide equação 4.6):

$$\overline{PA} = \frac{\sum_{i=1}^{i=n} PA_n}{n} \tag{4.6}$$

Essas métricas são especialmente úteis para verificar a eficácia geral do plantio e o potencial de produção.

Ambos os algoritmos resultam em imagens anotadas que fornecem uma representação visual clara das falhas no plantio e das área cultivada. Além disso, os resultados quantitativos são registrados para análise posterior, permitindo comparações entre diferentes áreas de plantio ou avaliação de mudanças na plantação ao longo do tempo.

5 Resultados

Nesse capítulo, serão apresentados os resultados obtidos pela confecção desse projeto, tal como as análises pertinentes aos mesmos. Os resultados consistem em gráficos, tabelas e imagens, todos obtidos através de funções disponibilizadas nos *frameworks* ou por códigos autorais. As seções desse capítulo foram segregadas de modo a manter a sequência dos procedimentos realizados no capítulo 4. Demais resultados acerca da rede que obteve o melhor desempenho, além de exemplos de *batches* de imagens utilizados durante o treino, validação e teste podem ser visualizados no Apêndice A deste documento.

5.1 Dados Sintéticos

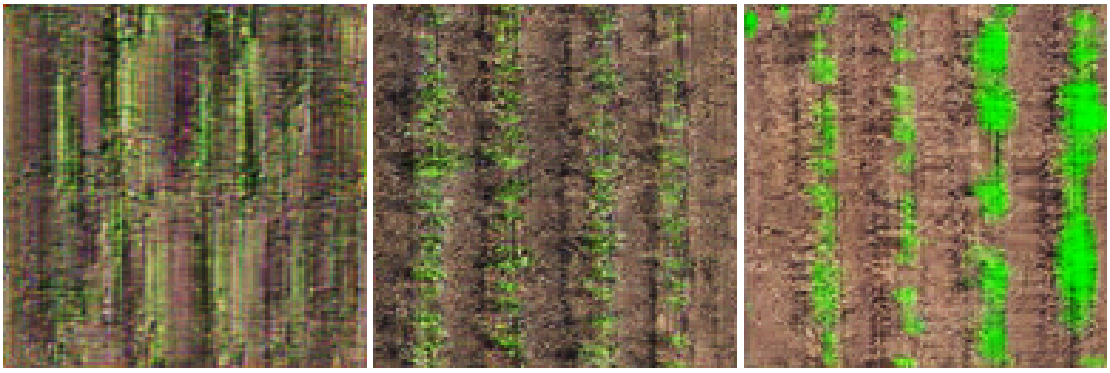
Nessa seção, as imagens sintéticas geradas pela GAN nas três resoluções (64×64, 128×128 e 256×256) serão fornecidas. Para uma percepção mais generalista de todos os resultados obtidos, serão mostradas, para cada resolução, imagens na parte inicial, medial e final do treinamento. Tais resultados são sumarizados nas figuras 5.27, 5.28 e 5.29.

(a) Época n° 100 (b) Época n° 600 (c) Época n° 611

Figura 5.27 – Exemplos de imagens sintéticas com resolução 64×64

Fonte: autores

Para a época de número 100, percebe-se que o modelo começa a aprender a mimetizar as lacunas verdes, o que se concretiza em um bom resultado na época de número 600. A partir de época 611, o sistema começa a repetir padrões indesejados.

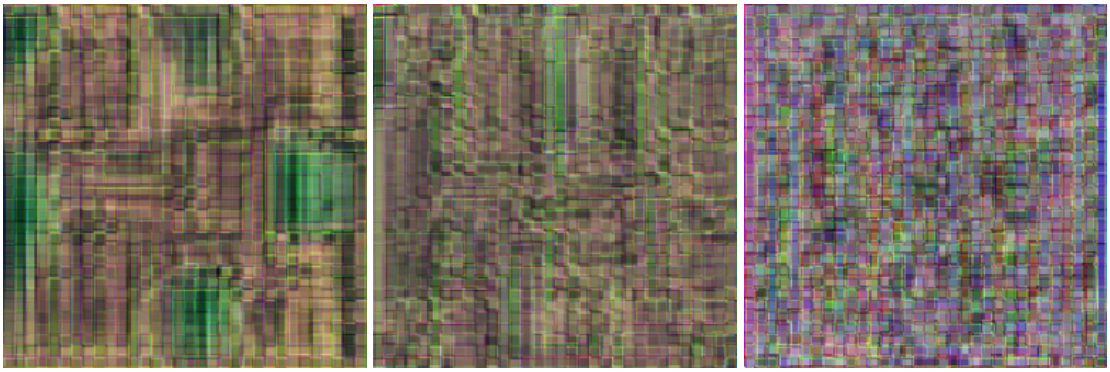


(a) Época n° 0 (b) Época n° 610 (c) Época n° 952

Figura 5.28 – Exemplos de imagens sintéticas com resolução 128×128

Fonte: autores

Já para a resolução 128×128, o modelo já consegue produzir imagens relativamente boas a partir da primeira época de execução. Por volta da metade do treinamento, começa a reproduzir as melhores imagens que obtivemos no projeto, na época 610. Por fim, o modelo começa a novamente reproduzir resultados indesejados, como a saturação muito forte do verde na época 952.



(a) Época n° 150 (b) Época n° 200 (c) Época n° 330

Figura 5.29 – Exemplos de imagens sintéticas com resolução 256×256

Fonte: autores

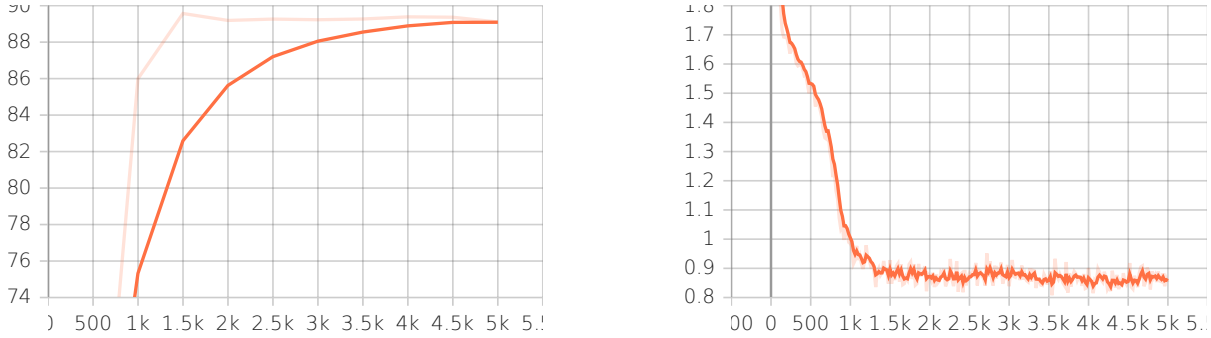
Em relação à maior resolução utilizada, o sistema não conseguiu, em nenhuma época, resultados satisfatórios. Apesar de, na época de número 200, o sistema esboçar certa melhora em relação à fase inicial, tal melhoria não é concretizada em épocas futuras, gerando resultados ininteligíveis. Para as três resoluções, o modelo apresentou um bom potencial de mimetização na fase inicial, esboçou os melhores resultados no meio do treinamento e, ao fim do mesmo, apresentou declínio em relação à compreensibilidade dos resultados. Dessa maneira, as únicas imagens incorporadas no conjunto de dados de treinamento se referem às produzidas da época 610 com resolução de 128×128. Vale ressaltar que estas imagens serão usadas apenas para fim de comparação no último modelo testado.

5.2 Análise e Comparação das Arquiteturas

Nos seguintes tópicos, serão apresentados os valores obtidos conforme as métricas apresentadas em 4.2.1. Vale ressaltar que o AP mostrado se refere ao AP50, que leva em consideração apenas *bounding boxes* previstas com IoU maior do que 50%. Os gráficos de progressão do AP e do *loss* foram obtidos, em todos os casos, com o auxílio do TensorBoard, ferramenta de fácil visualização de dados de treinamento em algoritmos de ML. Nas análises subsequentes, é importante diferenciar AP de validação e AP de teste. Enquanto o primeiro é calculado durante o treinamento e tem sua curva esboçada de acordo com o conjunto de imagens de validação, o segundo é calculado utilizando apenas o modelo final com o conjunto de dados de teste.

5.2.1 Desempenho das Arquiteturas do Detectron2 Testadas

Os gráficos a seguir representam a evolução do AP50 e da *loss function* durante todo o período de treinamento de 5 mil épocas. As Figuras 5.30 e 5.31 representam os gráficos obtidos pela arquitetura *Faster R-CNN* com ResNet de profundidade de 50 e 101 camadas, respectivamente. Apesar da curva original de AP estar presente na figura (laranja fraco), a mesma passa por um processo de *smoothing* para facilitar a visualização (laranja forte).

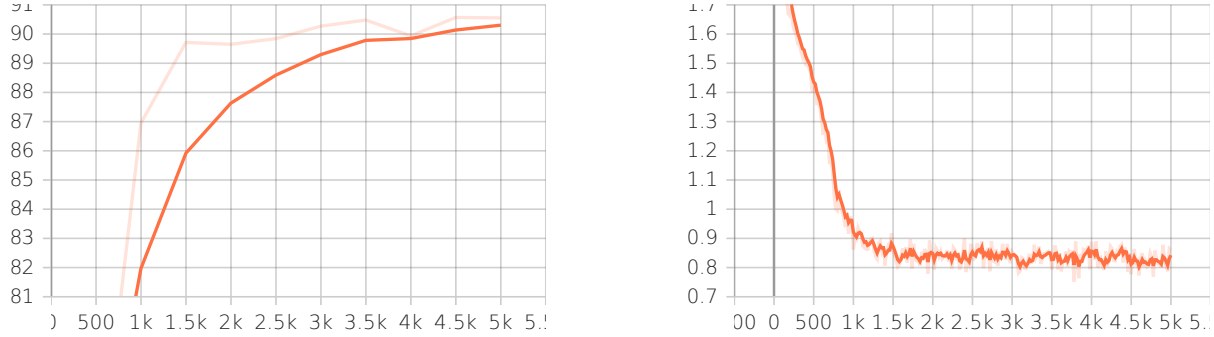


(a) AP

(b) Loss

Figura 5.30 – *Faster R-CNN*-R50

Fonte: autores



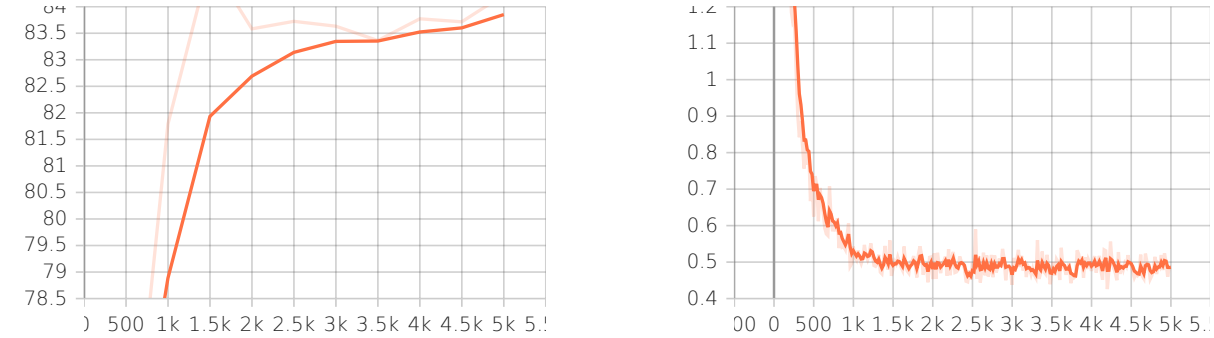
(a) AP (b) Loss

Figura 5.31 – *Faster R-CNN-R101*

Fonte: autores

Note que a curva do AP da *Faster R-CNN-101* possui um desempenho levemente superior à *Faster R-CNN-R50*. Isso pode ser justificado pela maior profundidade do *backbone* da *Faster R-CNN-R101*, possibilitando melhor extração das características.

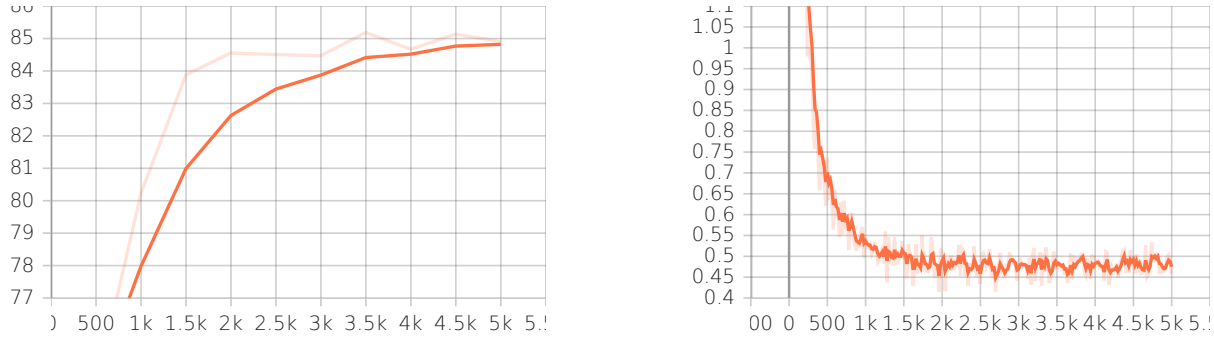
Outrossim, estão presentes nas Figuras 5.32 e 5.33 os gráficos obtidos pelas arquiteturas *RetinaNet-R50* e *RetinaNet-R101*, respectivamente.



(a) AP (b) Loss

Figura 5.32 – *RetinaNet-R50*

Fonte: autores



(a) AP (b) Loss

Figura 5.33 – *RetinaNet-R101*

Fonte: autores

Novamente, note que a arquitetura com *backbone* mais profunda desempenhou melhor quando validada no processo de treinamento, indicando melhor extração de características da imagem. Entretanto, apesar da *RetinaNet* ser considerada um modelo mais avançado do que a *Faster R-CNN* para detecção de objetos, a mesma apresentou pior desempenho durante o treinamento. Existem algumas possíveis causas que justificam esse efeito:

- os hiperparâmetros adotados como base e aplicados à ambas arquiteturas podem ter favorecido a *Faster R-CNN* para esse *dataset* em específico;
- fatores aleatórios: as redes possuem componentes estocásticos, o que também pode justificar pior desempenho para o modelo que, na teoria, é mais avançado.

Por fim, na Tabela 5.6 temos o AP calculado no conjunto de dados de teste para todos os modelos do Detectron2.

Modelo	**AP**
Faster R-CNN-R50	84.64
Faster R-CNN-R101	83.34
RetinaNet-R50	50.53
RetinaNet-R101	48.45

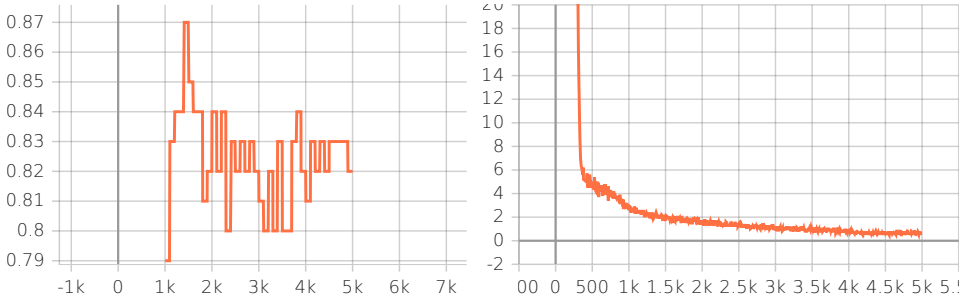
Tabela 5.6 – Tabela comparativa do AP de teste dos modelos

Perceba que, novamente, a *Faster R-CNN* desempenhou melhor do que a RetinaNet. Os valores obtidos no teste foram nitidamente inferiores, principalmente se tratando da RetinaNet, do que os valores obtidos na etapa de validação durante o treinamento. Isso é explicado pelo fato de que o conjunto de dados de validação é diferente do conjunto de dados de teste, implicando a possibilidade das imagens presentes em cada conjunto possuírem características ligeiramente diferentes. Além disso, é razoável pressupor que o modelo pode ter sofrido um grau de *overfitting*, fazendo com que sua performance piore para casos em que não havia processado anteriormente.

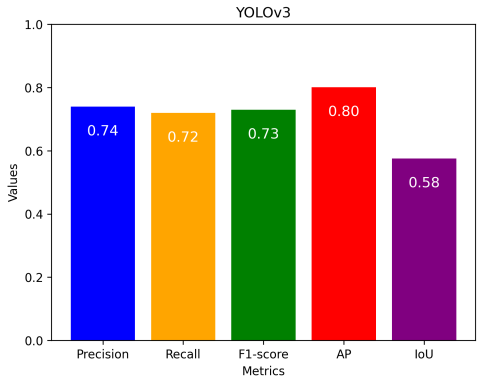
5.2.2 Desempenho das Arquiteturas YOLO Testadas

Nesta seção, apresentamos uma análise do desempenho das diferentes arquiteturas YOLO testadas.

Em primeira mão, vamos analisar os resultados da YOLOv3, YOLOv4 e YOLOv7, obtidos pelo *framework* Darknet e ilustrados nas Figuras 5.34, 5.35 e 5.36, respectivamente.



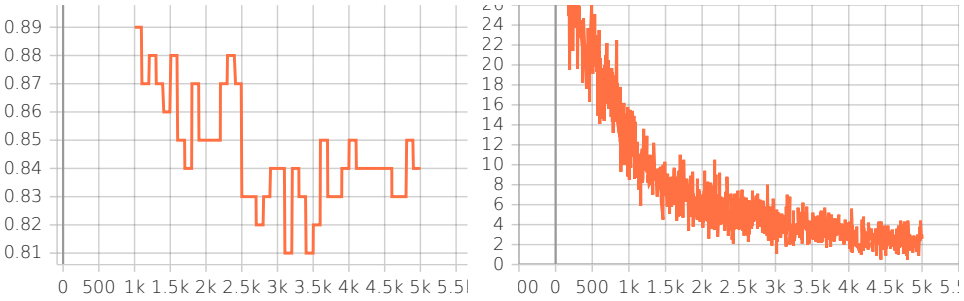
(a) AP durante treinamento (b) Loss durante treinamento



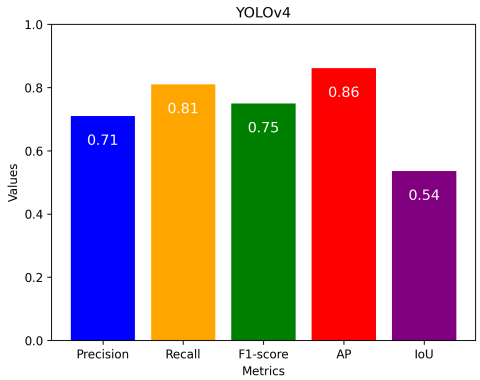
(c) Métricas no teste da rede

Figura 5.34 – Resultados de treinamento e teste para YOLOv3

Fonte: autores



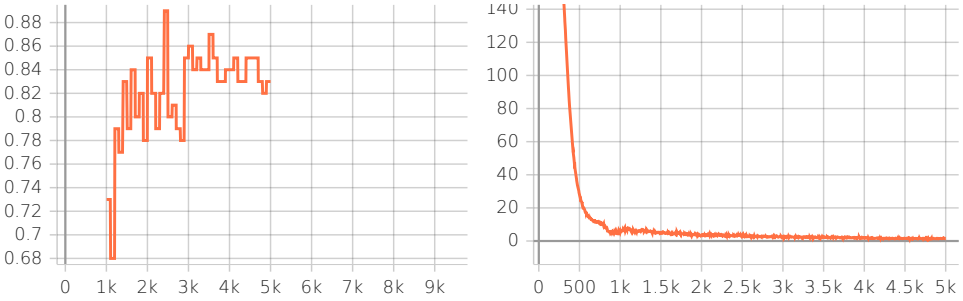
(a) AP durante treinamento (b) Loss durante treinamento



(c) Métricas no teste da rede

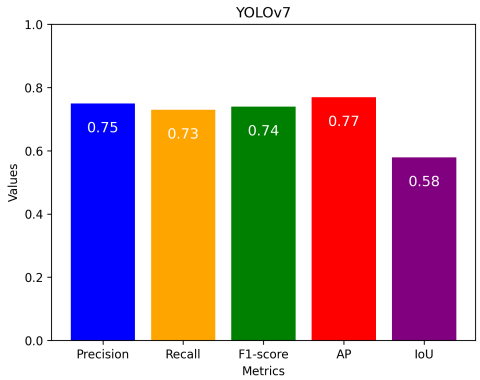
Figura 5.35 – Resultados de treinamento e teste para YOLOv4

Fonte: autores



(a) AP durante treinamento

(b) Loss durante treinamento



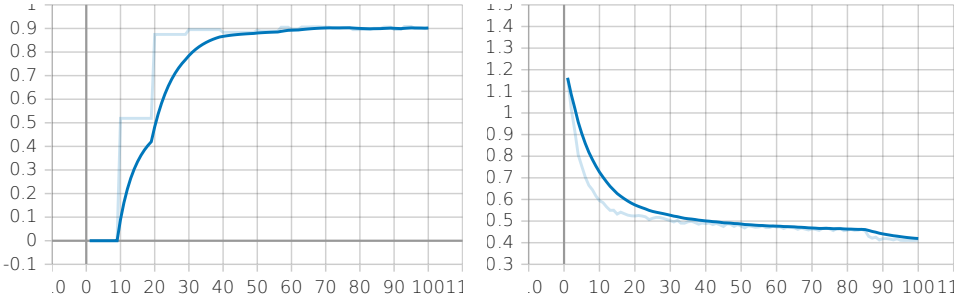
(c) Métricas no teste da rede

Figura 5.36 – Resultados de treinamento e teste para YOLOv7

Fonte: autores

Em relação às curvas de AP, é válido salientar que nas YOLOv3 e YOLOv4 não é possível perceber uma evolução em virtude do fato de que o AP é somente calculado após a milésima iteração. Além disso, para esses modelos, a involução dessa mesma métrica pode ser justificada por um provável *overfitting*. Para as curvas de *loss*, é importante observar que principalmente nas arquiteturas YOLOv3 e YOLOv7, ocorre uma estagnação muito próxima de zero, indicando bom desempenho das arquiteturas. Por fim, em relação às métricas validadas com o conjunto de teste, percebe-se maior *recall*, *F1-score* e AP na YOLOv4 e melhor precisão e IoU por parte da YOLOv7, essa última métrica empatada com a YOLOv3.

Para o modelo YOLOv6m (desenvolvido pela Meituan), a Figura 5.37 corresponde aos gráficos de AP e *loss* durante o treinamento.



(a) AP durante treinamento (b) Loss durante treinamento

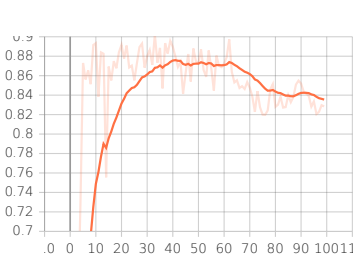
Figura 5.37 – Resultados de treinamento para YOLOv6m

Fonte: autores

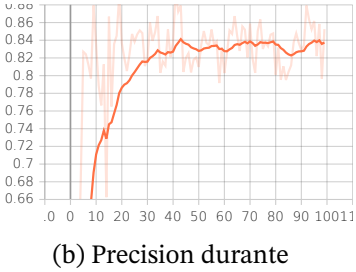
Como pode ser observado no gráfico em 5.37a, o desempenho da rede em relação ao AP durante o treinamento foi superior aos outros modelos apresentados até agora (89% havia sido o máximo atingido pelos modelos Darknet, como pode ser observado no gráfico em 5.35a). O AP evoluiu de tal maneira que chegou até 90%, mostrando um resultado promissor em relação a este modelo. Porém, a curva de *loss* em 5.37b sugere uma dificuldade do modelo em minimizar o erro entre as previsões e as marcações verdadeiras, pois o modelo parece estar convergindo e a partir do valor de *loss* entre 0.5 e 0.4, não observa-se uma mudança significativa.

Ao testar o modelo nos dados de teste, obteve-se um valor de AP de 0.784, o que mostra uma diferença de aproximadamente 12% com o que foi obtido durante o treinamento. Este fato sugere que pode haver algum nível de *overfitting* durante o treinamento. Isso significa que, para novos dados que o modelo nunca viu antes, ele pode não desempenhar da maneira esperada.

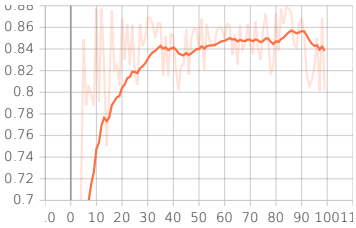
Em seguida, as Figuras 5.38 e 5.39 se referem aos resultados das arquiteturas YOLOv5m e YOLOv8m, todos obtidos pelo *framework* da Ultralytics.



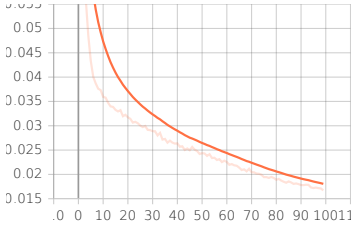
(a) AP durante treinamento



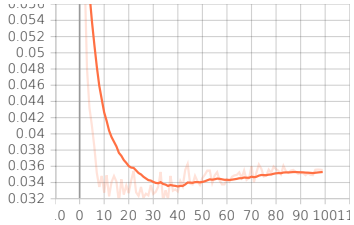
(b) Precision durante treinamento



(c) Recall durante treinamento



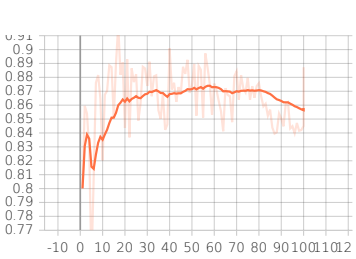
(d) Loss durante treinamento



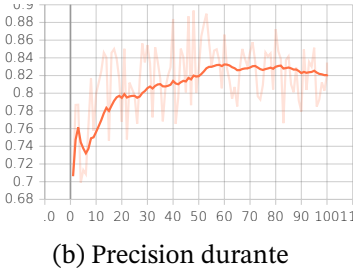
(e) Loss durante validação

Figura 5.38 – Resultados de treinamento para YOLOv5m

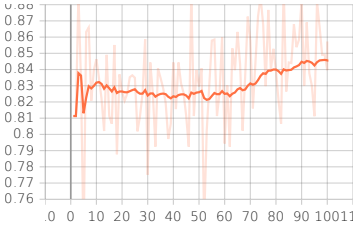
Fonte: autores



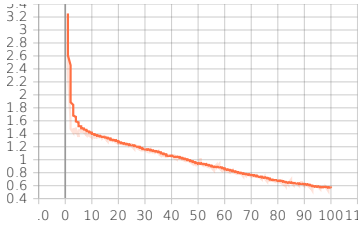
(a) AP durante treinamento



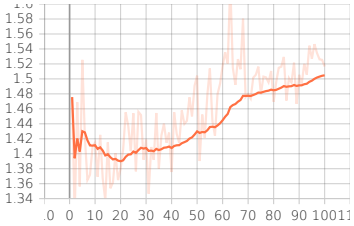
(b) Precision durante treinamento



(c) Recall durante treinamento



(d) Loss durante treinamento



(e) Loss durante validação

Figura 5.39 – Resultados de treinamento para YOLOv8m

Fonte: autores

Em relação ao AP durante o treinamento, é possível observar que ambas as curvas possuem fases de evolução, estagnação e involução, respectivamente. Novamente, o *overfitting* é o principal candidato a ser a causa das duas últimas fases. Percebe-se também forte semelhança nas curvas de precisão, ambas apresentando forte evolução. Se tratando do *recall*, percebe-se melhor evolução na YOLOv5m, indicando que a mesma conseguiu desempenhar melhor na detecção de TPs. É cabível salientar, também, forte semelhança entre as curvas de *loss* durante o treinamento. Por fim, diferentemente da YOLOv5m, a YOLOv8m

apresenta um aumento significativo do *loss* durante a validação, sendo um indicativo de um aprendizado não tão eficiente por parte da YOLOv8m.

Após a análise detalhada de cada modelo YOLO testado, a Tabela 5.7 apresenta o valor de AP no conjunto de teste para cada um deles.

Modelo	**AP**
YOLOv3	80
YOLOv4	86
YOLOv5m	82.7
YOLOv6m	78.4
YOLOv7	77
YOLOv8m	84.5

Tabela 5.7 – Tabela comparativa do AP de teste dos modelos YOLO

Fica evidente que os modelos YOLOv4 e YOLOv8m apresentam ótimos resultados, diferentemente dos modelos YOLOv6m e YOLOv7 que mostraram um desempenho inferior. Um aspecto interessante que pode ser observado é que não necessariamente modelos mais recentes da arquitetura YOLO obtiveram um desempenho melhor para o *dataset* utilizado. Isso mostra que cada conjunto de dados apresenta diferentes desafios e complexidades, além de ressaltar a importância de realizar experimentos com diferentes arquiteturas e versões.

5.2.3 Comparação Geral entre os Modelos

Após a obtenção e análise dos modelos supracitados, faz-se necessária uma comparação mais direta entre os resultados obtidos por tais. Em primeira instância, na figura 5.40, compararemos o AP de todos os modelos utilizados. Em seguida, para uma mesma imagem do conjunto de dados de teste, disponibilizaremos a imagem após a inferência de cada modelo, com caixas delimitadores e seus respectivos índice de confiança.

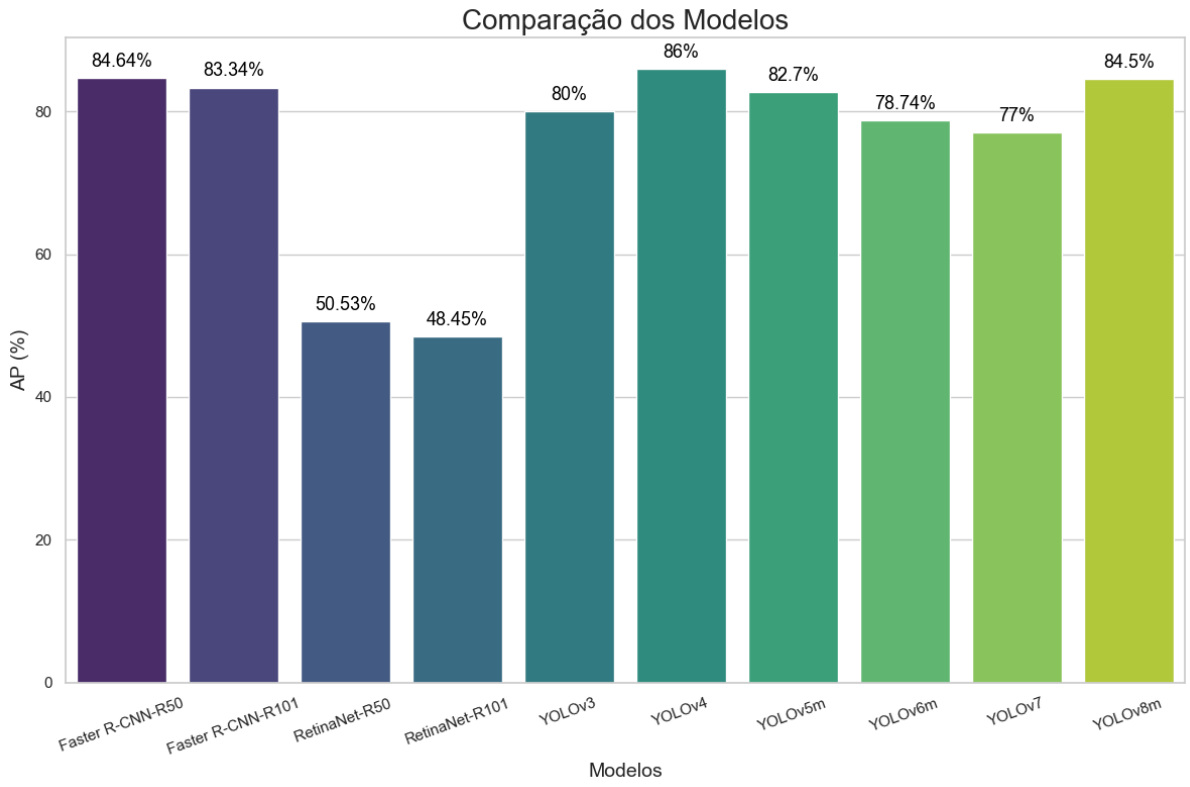
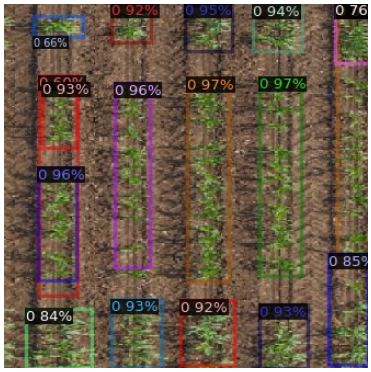


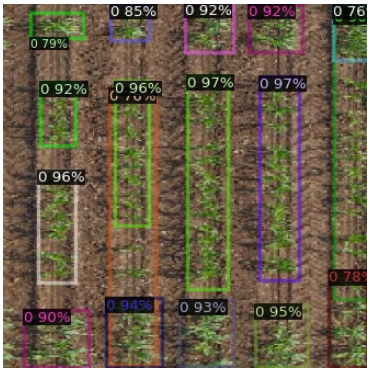
Figura 5.40 – Comparação do AP dos Modelos nos dados de teste

Fonte: autores

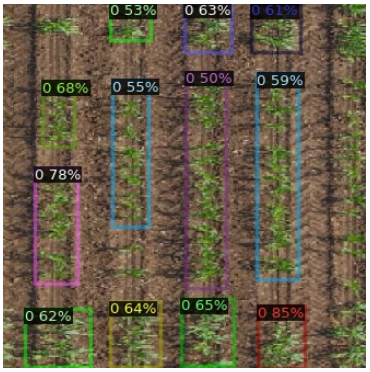
Para a comparação da inferência dos modelos, optou-se pela escolha de uma foto que possuísse uma boa variação nas características das lavouras, como linhas de plantio longas, curtas e apresentando falhas. Essa decisão facilita a diferenciação do desempenho dos modelos.



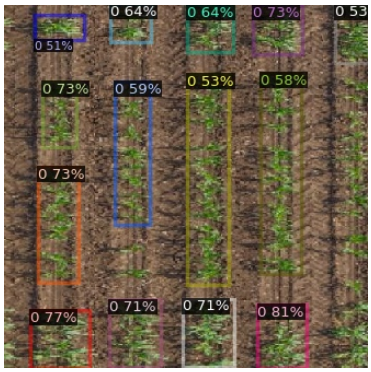
(a) *Faster R-CNN*-R50



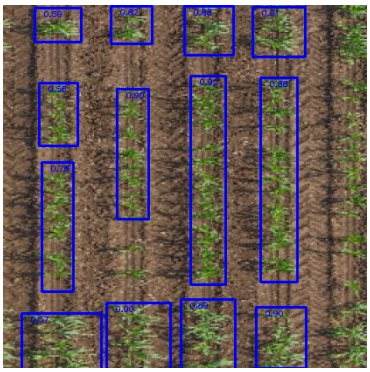
(b) *Faster R-CNN*-R101



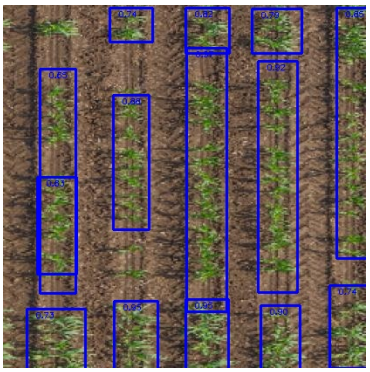
(c) *RetinaNet-R50*



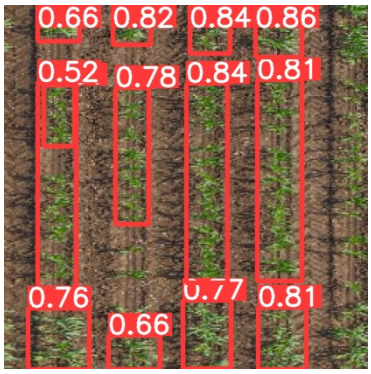
(d) *RetinaNet-R101*



(e) YOLOv3

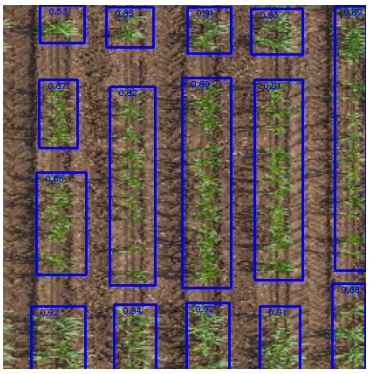


(f) YOLOv4



(g) YOLOv5m

(h) YOLOv6m



(i) YOLOv7

(j) YOLOv8

Figura 5.41 – Inferências dos modelos

Fonte: autores

5.3 Otimização e Desempenho do Modelo Selecionado (YOLOv8)

Como já discutido, a profundidade da rede pode afetar diretamente na performance da mesma. Nesse sentido, serão apresentados os resultados dos diferentes tamanhos da YOLOv8 testados: *nano* (YOLOv8n), *small* (YOLOv8s), *medium* (YOLOv8m), *large* (YOLOv8l) e *extra large* (YOLOv8x).

Em seguida, os resultados da otimização dos hiperparâmetros da rede de melhor desempenho serão apresentados, de forma a ilustrar as melhorias da rede ao alterar sua configuração.

5.3.1 Resultados após Testes de Profundidades

Primeiramente, vamos analisar a evolução do AP de treinamento de cada modelo na Figura 5.42.

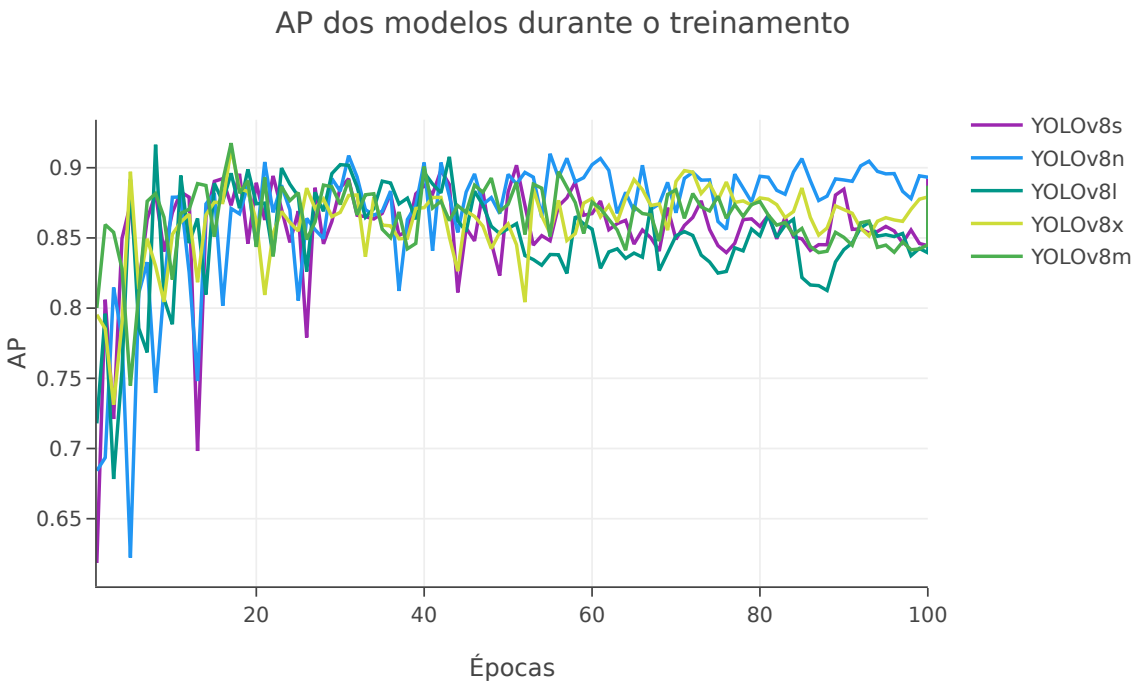


Figura 5.42 – Comparação do AP no treinamento entre os diferentes tamanhos do modelo

Fonte: autores

Surpreendentemente, o modelo mais raso apresentou melhor AP durante o treinamento. Entretanto, tal resultado não é conclusivo para definição do modelo ideal, visto que

esse só poderá ser selecionado após a visualização dos APs de teste, os quais serão analisados na Figura 5.43.

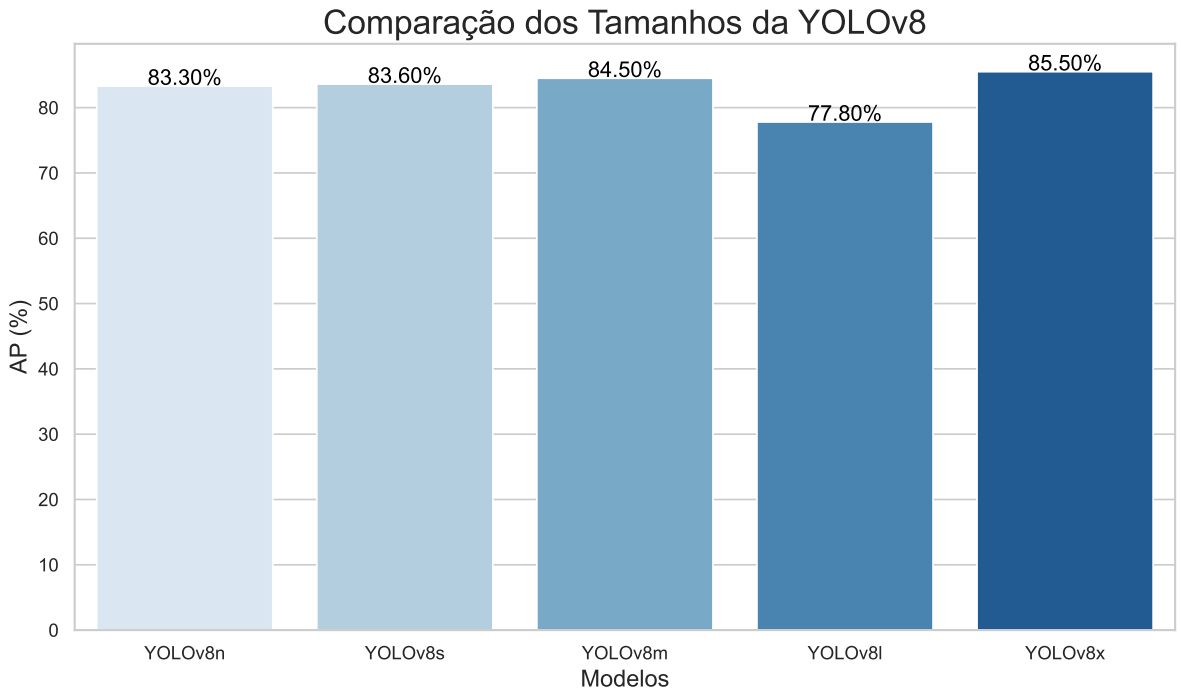


Figura 5.43 – Comparação do AP nos dados de teste entre os diferentes tamanhos do modelo

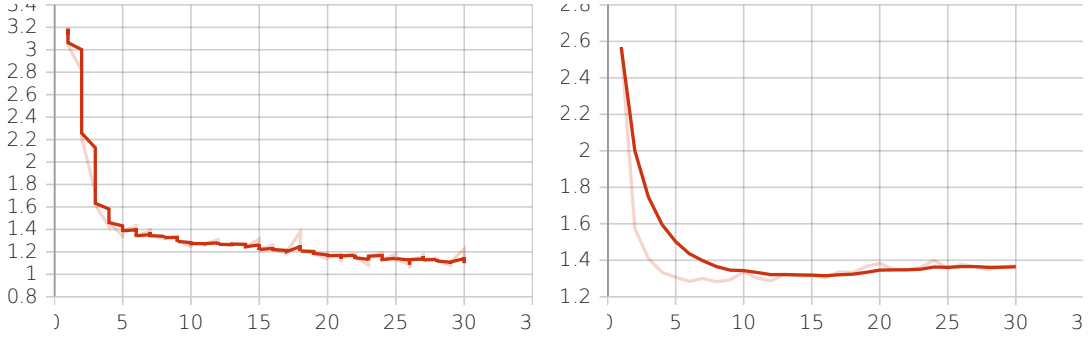
Fonte: autores

Como pode ser observado, o modelo mais profundo possui melhor desempenho de AP no teste, provavelmente em virtude da melhor detecção de características.

5.3.2 Resultados após Ajuste de Hiperparâmetros

Os resultados obtidos após o ajuste dos hiperparâmetros do modelo da YOLOv8x serão apresentados a seguir (a configuração com os melhores hiperparâmetros pode ser visualizada na Tabela 4.5).

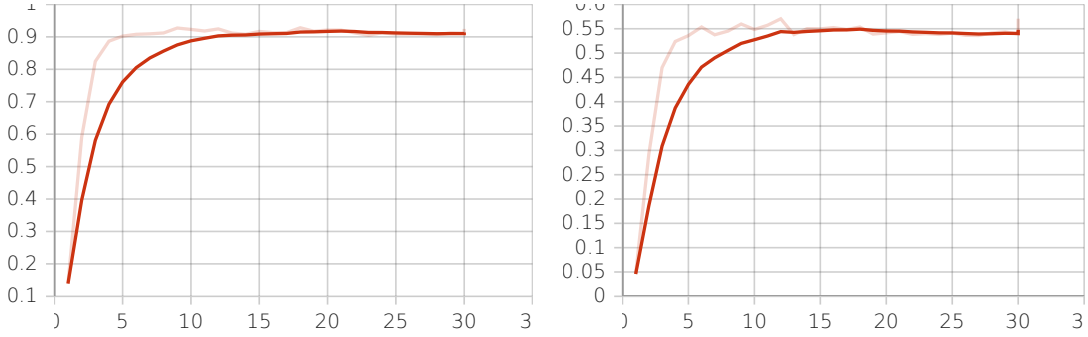
Para o treinamento, as Figuras 5.44, 5.45 e 5.46 apresentam, respectivamente, a *Loss* durante o treinamento e a validação, o AP para um IoU de 50% e um IoU variando de 50% a 95%, além da *precision* e *recall*.



(a) Loss durante o treinamento (b) Loss durante a validação

Figura 5.44 – Gráficos de Loss durante o treinamento e validação

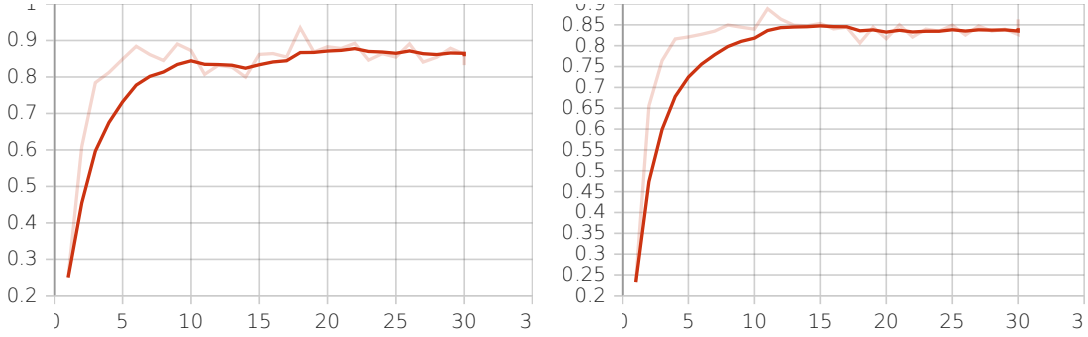
Fonte: autores



(a) AP para IoU de 50% (b) AP para IoU variando de 50% a 95%

Figura 5.45 – Gráficos de AP para diferentes valores de IoU

Fonte: autores



(a) Precision (b) Recall

Figura 5.46 – Gráficos de *Precision* e *Recall*

Fonte: autores

Para a fase de teste, a Tabela 5.8 mostra os valores obtidos para diferentes métricas.

Métrica	**Valor**
Precision	0.804
Recall	0.817
AP50	0.892
AP50-95	0.544

Tabela 5.8 – Valores das métricas nos dados de teste para o modelo YOLOv8x otimizado

Por último, a Figura 5.47 apresenta um conjunto de imagens dos dados de teste que obtiveram uma ótima detecção das linhas de plantação de sorgo, enquanto a Figura 5.48 mostra imagens em que a rede obteve um desempenho menor em identificar as linhas. Para uma melhor visualização da localidade e alinhamento das linhas de plantação, a Figura 5.49 contém imagens que ao invés das *bounding boxes* contém apenas uma linha em cima da plantação. Isso também é interessante para futuras aplicações que precisem apenas do referencial da linha e não necessita de informações como a densidade da plantação, como por exemplo para guiar veículos autônomos.

Figura 5.47 – Exemplos de imagens com boa detecção

Fonte: autores

Figura 5.48 – Exemplos de imagens com detecções faltando

Fonte: autores

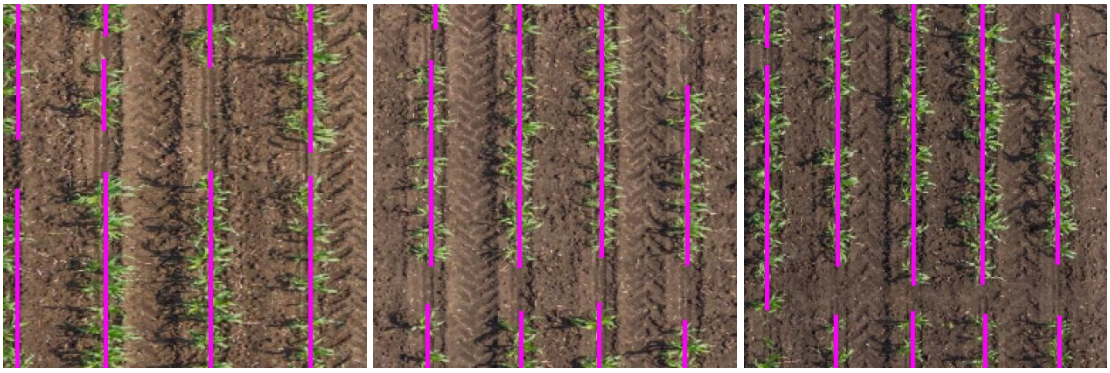


Figura 5.49 – Exemplos de imagens com linhas ao invés de *bounding boxes*

Fonte: autores

5.3.3 Incorporação dos Dados Sintéticos na Base de Treinamento

Nessa seção, serão apresentados os resultados obtidos pelo modelo final da YOLOv8x treinada com a base contendo dados sintéticos, de forma a verificar se ocorre melhoria. Para melhor validação da compreensibilidade das imagens geradas pela GAN, as mesmas serão inferidas pelo modelo apresentado anteriormente, como ilustrado na Figura 5.50.

Figura 5.50 – Exemplos de inferência nas imagens sintéticas

Fonte: autores

Além disso, para verificar se a nova base fornece um melhor conjunto de dados de treinamento, os mesmos testes obtidos na Tabela 5.8 serão replicados, como mostra na Tabela 5.9.

Métrica	**Valor**
Precision	0.74
Recall	0.833
AP50	0.852
AP50-95	0.518

Tabela 5.9 – Valores das métricas nos dados de teste utilizando dados sintéticos

Apesar de a incorporação de imagens sintéticos no conjunto de imagens de treinamento ter acarretado em um melhor *recall*, todas as outras métricas observadas indicam pior desempenho. Um dos fatores que podem ter causado esse efeito é a diferença relativamente grande de resolução, exigindo um *resize* brusco para possibilitar o processamento, o que gera grandes distorções nas imagens.

5.4 Cálculo da Área Cultivada e das Falhas

No presente tópico, apresentaremos os resultados obtidos através do cálculo do percentual da área cultivada e a da extensão das falhas na plantação.

Iniciamos com uma análise visual de imagens selecionadas do nosso conjunto de dados de teste, ilustradas na Figura 5.51. As imagens apresentam tanto o aproveitamento do solo quanto o tamanho total das falhas na plantação, apresentadas em pixels. Em virtude das fotos obtidas pelo *dataset* disponibilizado possuírem pequenas distorções em algumas fotos, o cálculo dos tamanhos das falhas em metros fica impossibilitado.

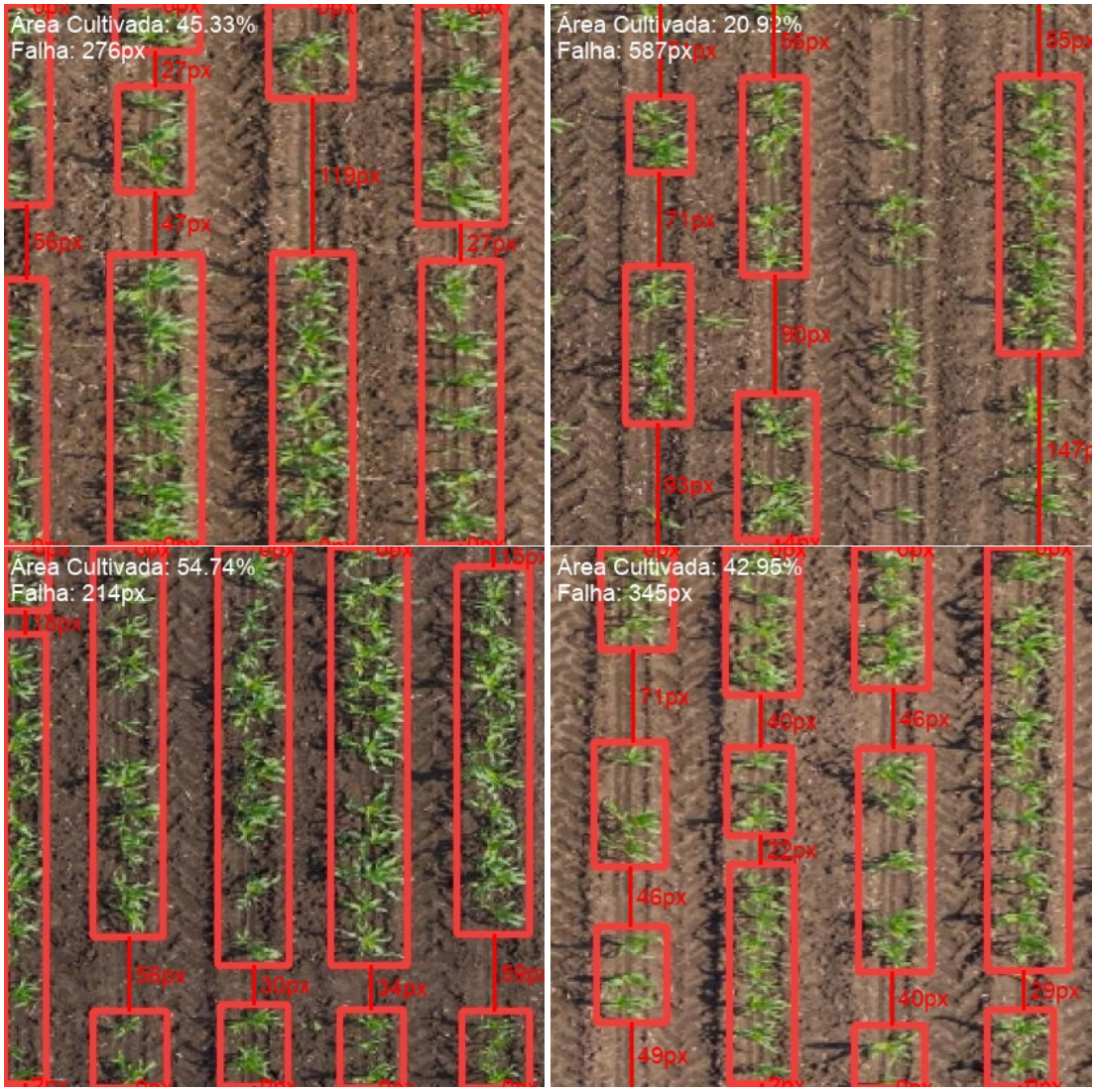


Figura 5.51 – Resultados do cálculo da área cultivada e das falhas na plantação

Fonte: autores

É crucial enfatizar que o cálculo preciso da área cultivada e das falhas depende, em grande medida, da correta detecção das linhas de plantação pelo modelo.

Seguindo para uma análise mais quantitativa, foi calculado o coeficiente de correlação de Pearson (SCRIBBR, 2022) entre as variáveis em questão. Descobrimos um coeficiente de -0.579, que aponta para uma correlação negativa moderada entre a área cultivada e as falhas detectadas na plantação. Este resultado pode ser visualizado no gráfico da Figura 5.52.

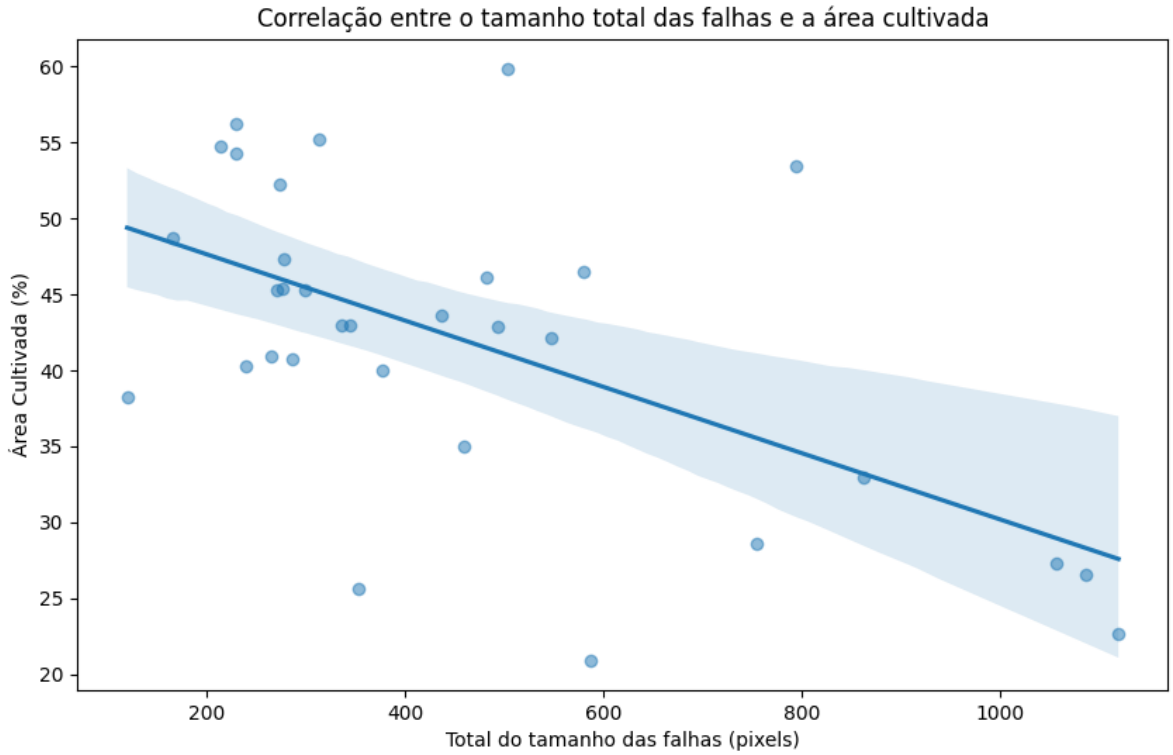


Figura 5.52 – Gráfico demonstrando a correlação negativa moderada entre a área cultivada e as falhas na plantação

Fonte: autores

É importante ressaltar que esse é um comportamento esperado do sistema, uma vez que quanto maior o tamanho das falhas, em pixels, espera-se menor percentual de solo cultivado.

Por último, realizamos um cálculo abrangente da área cultivada média e a somatória dos tamanhos das falhas utilizando todas as 32 imagens disponíveis em nosso conjunto de dados de teste. Este processo nos permite simular um mapeamento da plantação de forma mais realista, obtendo uma representação prática da aplicação deste trabalho. Os valores encontrados estão na Tabela 5.10.

Área Cultivada Média (%)	**Somatória Total das Falhas (px)**
42.02	14634

Tabela 5.10 – Área cultivada média e somatória total das falhas.

6 Conclusões

Este trabalho apresenta uma abordagem para a detecção de linhas de plantio, cálculo do percentual de área cultivada e identificação de falhas em imagens aéreas de lavouras de sorgo. Nesse sentido, foram estudadas diversos modelos de redes neurais convolucionais para detecção de objetos, de modo a treiná-los com nosso *dataset* e validar os respectivos resultados. De mesmo modo, foram estudadas, também, outras técnicas relevantes para o contexto de IA, como a utilização de *data augmentation*, para amplificar a quantidade de dados de treinamento; *transfer learning* para a utilização de redes pré-treinadas em outros *datasets*; e, por fim,a geração de dados sintéticos na tentativa de obter um modelo mais generalista.

Para a comparação direta entre as arquiteturas escolhidas (*RetinaNe*t, *Faster R-CNN* e YOLO) e seus respectivos modelos, optamos pelo treinamento com um conjunto de hiperparâmetros com menos variações possíveis, de modo a prosseguir com a melhor arquitetura levando em consideração o AP, documentação, facilidade de uso e de obtenção de resultados. Nesse sentido a YOLOv8 mostrou-se a melhor opção. Em seguida, diferentes profundidades desse mesmo modelo foram testadas de modo a verificar se uma detecção mais profunda das características ocasionaria, consequentemente, em um melhor resultado. Tal hipótese mostrou-se válida, assim como observado na Figura 5.43, onde mostra uma tendência de melhora do AP conforme a profundidade aumenta. Apesar da YOLOv8l não ter apresentado melhora, não é o bastante para refutar a tese.

Além disso, foi feita a otimização do modelo escolhido, o YOLOv8x, variando os hiperparâmetros, o que mostrou-se uma proposta sucessora, visto que aumentou o AP em 4%, resultando nos valores de *precision*, *recall*, AP50 e AP50-95 de 0.804, 0.817, 0.892 e 0.544, respectivamente. Em relação à captação das falhas e do índice de aproveitamento do solo, pode-se inferir que os mesmos foram proveitosos, vistos que fornecem fortes noções de monitoramento do plantio.

A técnica de *data augmentation* foi implementada para aprimorar o desempenho dos modelos em face de possíveis alterações nas imagens presentes no *dataset*. O uso de *synthetic data* também foi explorado para aumentar o *dataset* original, em que as imagens sintéticas geradas foram satisfatórias a princípio. Porém, ao introduzi-las na base de dados de treinamento do melhor modelo, os resultados produzidos foram inferiores quando comparados com o treinamento sem utilizar os dados sintéticos.

6.1 Limitações e Desafios

Apesar dos resultados significativos que conseguimos alcançar com nosso trabalho, enfrentamos alguns desafios. Um dos principais obstáculos foi a necessidade de uma detecção precisa das linhas de plantio para calcular corretamente a área cultivada e as falhas nas linhas de plantio. Isso exigiu um alto nível de precisão do nosso modelo, o que é desafiador alcançar, especialmente considerando as limitações de nosso *dataset*.

Em segunda análise, ressalta-se a limitação do primeiro *dataset* disponibilizado para a realização do projeto. Como já citado nesse documento, o mesmo contém pouquíssimas imagens que, mesmo com *data augmentation*, enviesaria a capacidade de generalização do modelo. Além disso, para esse mesmo *dataset*, também havia uma grande dificuldade de rotulagem, visto que muitas imagens eram carentes de características rotuláveis.

O segundo e definitivo *dataset* também oferecia pouco controle da qualidade das imagens, visto que a princípio fornecia 50.000 imagens ocasionalmente distorcidas. Tais aspectos fizeram com que optássemos pela seleção manual de 298 imagens para posterior *data augmentation*. Também tivemos que nos restringir a limitação de linhas de plantio com orientações fixas, visto que a detecção com caixas delimitadores que se ajustem ao ângulo faria mais sentido em um trabalho posterior. Outro fator impeditivo foi a grande quantidade de mudança de resolução das imagens, impedindo o cálculo das falhas em metros.

Além disso, em virtude da quantidade de modelos e versões de detecção de objetos diferentes, é cabível pontuar a dificuldade de familiarização com os respectivos *frameworks*, os quais, em algumas ocasiões, não eram tão bem documentados.

Por fim, embora tenhamos utilizado o *transfer learning* para acelerar e tornar mais eficientes nossos treinamentos, a limitação de recursos computacionais, de tempo e de um *dataset* com mais imagens e de maior qualidade nos impediu de explorar diferentes abordagens que poderiam ter produzido resultados ainda melhores, como por exemplo *ensemble learning* ou realizar o treinamento sem usar modelos pré-treinados. A grande requisição de capacidade computacional também fez com que precisássemos assinar uma plataforma de computação em nuvem para a realização dos treinamentos e desenvolvimento mútuo dos autores, o *Google Colab Pro*.

6.2 Perspectivas Futuras

Este trabalho abre caminho para uma série de possibilidades futuras na área de detecção de linhas de plantio. Com base nos resultados obtidos, futuros trabalhos podem ser desenvolvidos para melhorar ainda mais a precisão e eficiência da detecção de linhas de plantio, da área cultivada e da detecção de falhas.

Um dos desenvolvimentos mais promissores seria a implementação de um sistema

embarcado para uso em drones, capaz de realizar o reconhecimento em tempo real levando em consideração o *trade-off* entre precisão e velocidade de inferência do modelo. Isso permitiria mapear grandes áreas de plantação de forma muito mais eficiente, economizando tempo e recursos. Além disso, embarcar tal sistema exigiria que o treinamento fosse realizado em cima de figuras rotacionadas, criando um sistema mais generalista onde as orientações das linhas de plantio detectadas poderiam ser usadas para ajudar a corrigir trajetórias de veículos autônomos, evitando o esmagamento de plantações e aumentando a eficiência das operações agrícolas.

Além disso, a utilização de um outro *dataset* também pode ser interessante para estudar diferentes tipos de culturas agrícolas e *features* que podem ser extraídas. A depender da quantidade de informações disponibilizadas pelo *dataset*, também seria possível calcular o tamanho das falhas em metros.

Em última análise, o trabalho apresentado aqui representa um estudo significativo para a agricultura de precisão, com potencial para transformar a eficiência e a produtividade das operações agrícolas. No entanto, ainda há muitos aspectos que devem ser melhorados, principalmente no que diz respeito a disponibilidade de grande quantidade de imagens de alta qualidade para serem utilizadas no treinamento dos modelos de detecção.

Referências

ABDU, M.; BATISTA, I.; CARRASCO, A.; BRUM, C. South Atlantic magnetic anomaly ionization: A review and a new focus on electrodynamic effects in the equatorial ionosphere. **Journal of Atmospheric and Solar-Terrestrial Physics**, v. 67, n. 17, p. 1643–1657, 2005. Space Geophysics. ISSN 1364-6826. DOI: https://doi.org/10.1016/j.jastp.2005.01.014. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1364682605001240>. Citado na p. 16.

ALEXEYAB. **Darknet: Open Source Neural Networks in C**. GitHub, 2021. https://github.com/AlexeyAB/darknet. [Acesso em: 14/04/2023]. Citado nas pp. 35, 54.

ANDREO, V. Remote Sensing and Geographic Information Systems in Precision Farming, out. 2013. Citado na p. 17.

BAH, M. D.; HAFIANE, A.; CANALS, R. Deep Learning with Unsupervised Data Labeling for Weed Detection in Line Crops in UAV Images. **Remote Sensing**, v. 10, n. 11, 2018. ISSN 2072-4292. DOI: 10.3390/rs10111690. Disponível em: <https://www.mdpi.com/2072-4292/10/11/1690>. Citado na p. 45.

BAH, M. D.; HAFIANE, A.; CANALS, R. CRowNet: Deep Network for Crop Row Detection in UAV Images. **IEEE Access**, v. 8, p. 5189–5200, 2020. DOI: 10.1109/ACCESS.2019.2960873. Citado na p. 45.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. **YOLOv4: Optimal Speed and Accuracy of Object Detection**. 2020. arXiv: 2004.10934 [cs.CV]. Citado nas pp. 33, 34.

BOSERUP, E. **The Conditions of Agricultural Growth: The Economics of Agrarian Change Under Population Pressure**. Aldine, 1965. Citado na p. 17.

CARDOSO, L. A. S.; FARIAS, P. R. S.; SOARES, J. A. C. Use of Unmanned Aerial Vehicle in Sugarcane Cultivation in Brazil: A Review. **Sugar Tech**, v. 24, n. 6, p. 1636–1648, dez. 2022. ISSN 0974-0740. DOI: 10.1007/s12355-022-01149-9. Disponível em: <https://doi.org/10.1007/s12355-022-01149-9>. Citado nas pp. 43, 44.

DAIGH, A. L. M.; DEJONG-HUGHES, J.; ACHARYA, U. Projections of yield losses and economic costs following deep wheel-traffic compaction during the 2019 harvest. **Agricultural & Environmental Letters**, v. 5, n. 1, e20013, 2020. DOI: https://doi.org/10.1002/ael2.20013. eprint: https://acsess.onlinelibrary.wiley.com/doi/pdf/10.1002/ael2.20013. Disponível em: <https://acsess.onlinelibrary.wiley.com/doi/abs/10.1002/ael2.20013>. Citado na p. 16.

DENG, J.; DONG, W.; SOCHER, R.; LI, L.-J.; LI, K.; FEI-FEI, L. Imagenet: A large-scale hierarchical image database. In: IEEE. 2009 IEEE conference on computer vision and pattern recognition. 2009. P. 248–255. Citado nas pp. 38, 42.

DUMOULIN, V.; VISIN, F. **A guide to convolution arithmetic for deep learning**. arXiv, 2016. DOI: 10.48550/ARXIV.1603.07285. Disponível em: <https://arxiv.org/abs/1603.07285>. Citado nas pp. 24, 25.

ELAIDOUNI, M. **Evaluating Object Detection Models: Guide to Performance Metrics**. 2021. https://manalelaidouni.github.io/Evaluating-Object-Detection-Models-Guide-to-Performance-Metrics.html. [Acesso em: 30/04/2023]. Citado na p. 53.

GAD, A. F. **Faster R-CNN Explained for Object Detection Tasks**. Paperspace, 2020. [Acesso em: 12/05/2023]. Disponível em: <https://blog.paperspace.com/faster-r-cnn-explained-object-detection/>. Citado nas pp. 28–30.

GONZALEZ, R.; WOODS, R. **Digital Image Processing**. Pearson, 2018. ISBN 9780133356724. Disponível em: <https://books.google.com.br/books?id=0F05vgAACAAJ>. Citado na p. 18.

GÜRSOY, S. Soil Compaction Due to Increased Machinery Intensity in Agricultural Production: Its Main Causes, Effects and Management. In: AHMAD, F.; SULTAN, M. (Ed.). **Technology in Agriculture**. Rijeka: IntechOpen, 2021. cap. 5. DOI: 10.5772/intechopen.98564. Disponível em: <https://doi.org/10.5772/intechopen.98564>. Citado na p. 16.

HAYKIN, S. **Redes Neurais: Princípios e Prática**. Bookman, 2001. Citado na p. 20.

HEYBURN, R.; BOND, R. R.; BLACK, M.; MULVENNA, M.; WALLACE, J.; RANKIN, D.; CLELAND, B. Machine learning using synthetic and real data: Similarity of evaluation metrics for different healthcare datasets and for different algorithms. In: DATA Science and Knowledge Engineering for Sensing Decision Support. WORLD SCIENTIFIC, jul. 2018. DOI: 10.1142/9789813273238_0160. Disponível em: <https://doi.org/10.1142/9789813273238_0160>. Citado na p. 40.

HO, N.; KIM, Y.-C. Evaluation of transfer learning in deep convolutional neural network models for cardiac short axis slice classification. **Scientific Reports**, v. 11, n. 1, p. 1839, jan. 2021. ISSN 2045-2322. DOI: 10.1038/s41598-021-81525-9. Disponível em: <https://doi.org/10.1038/s41598-021-81525-9>. Citado na p. 42.

HUSSAIN, F. **Your Handbook to Convolutional Neural Networks**. 2020. [Acesso em: 12/05/2023]. Disponível em: <https://medium.com/analytics-vidhya/your-handbook-to-convolutional-neural-networks-628782b68f7e>. Citado na p. 26.

JI, R.; QI, L. Crop-row detection algorithm based on Random Hough Transformation. **Mathematical and Computer Modelling**, v. 54, n. 3, p. 1016–1020, 2011. Mathematical and Computer Modeling in agriculture (CCTA 2010). ISSN 0895-7177. DOI: https://doi.org/10.1016/j.mcm.2010.11.030. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0895717710005212>. Citado na p. 44.

JOCHER, G.; CHAURASIA, A.; STOKEN, A.; BOROVEC, J.; NANOCODE012; KWON, Y.; MICHAEL, K.; TAOXIE; FANG, J.; IMYHXY; LORNA; YIFU, Z.; WONG, C.; V, A.; MONTES, D.; WANG, Z.; FATI, C.; NADAR, J.; LAUGHING; UNGLVKITDE; SONCK, V.; TKIANAI; YXNONG; SKALSKI, P.; HOGAN, A.; NAIR, D.; STROBEL, M.; JAIN, M. **ultralytics/yolov5: v7.0 - YOLOv5 SOTA Realtime Instance Segmentation**. Zenodo, nov. 2022. DOI: 10.5281/zenodo.7347926. Disponível em: <https://doi.org/10.5281/zenodo.7347926>. Citado na p. 34.

KOECH, K. E. **How Does Back-Propagation Work in Neural Networks? — towardsdatascience.com**. https://towardsdatascience.com/how-does-back-propagation-work-in-neural-networks-with-worked-example-bc59dfb97f48. [Acesso em: 29/05/2023]. Citado na p. 27.

KRÜGER, G.; SPRINGER, R.; LECHNER, W. Global Navigation Satellite Systems (GNSS). **Computers and Electronics in Agriculture**, v. 11, n. 1, p. 3–21, 1994. Global Positioning Systems in Agriculture. ISSN 0168-1699. DOI: https://doi.org/10.1016/0168-1699(94)90049-3. Disponível em: <https://www.sciencedirect.com/science/article/pii/0168169994900493>. Citado na p. 16.

LABELBOX. **Labelbox**. https://www.labelbox.com/. [Acesso em: 17/02/2023]. Citado na p. 49.

LI, C.; LI, L.; JIANG, H.; WENG, K.; GENG, Y.; LI, L.; KE, Z.; LI, Q.; CHENG, M.; NIE, W.; LI, Y.; ZHANG, B.; LIANG, Y.; ZHOU, L.; XU, X.; CHU, X.; WEI, X.; WEI, X. **YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications**. 2022. arXiv: 2209.02976 [cs.CV]. Citado nas pp. 36, 37.

LIN, T.-Y.; GOYAL, P.; GIRSHICK, R.; HE, K.; DOLLÁR, P. **Focal Loss for Dense Object Detection**. 2018. arXiv: 1708.02002 [cs.CV]. Citado nas pp. 30, 32.

LIN, T.; MAIRE, M.; BELONGIE, S. J.; BOURDEV, L. D.; GIRSHICK, R. B.; HAYS, J.; PERONA, P.; RAMANAN, D.; DOLL'A R, P.; ZITNICK, C. L. Microsoft COCO: Common Objects in Context. **CoRR**, abs/1405.0312, 2014. arXiv: 1405.0312. Disponível em: <http://arxiv.org/abs/1405.0312>. Citado nas pp. 38, 42.

LIU, X.; CHEN, S. W.; NARDARI, G. V.; QU, C.; OJEDA, F. C.; TAYLOR, C. J.; KUMAR, V. Challenges and Opportunities for Autonomous Micro-UAVs in Precision Agriculture.

IEEE Micro, v. 42, n. 1, p. 61–68, 2022. DOI: 10.1109/MM.2021.3134744. Citado na p. 18.

MAKTAB DAR OGHAZ, M.; RAZAAK, M.; KERDEGARI, H.; ARGYRIOU, V.; REMAGNINO, P. Scene and Environment Monitoring Using Aerial Imagery and Deep Learning. In: 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS). 2019. P. 362–369. DOI: 10.1109/DCOSS.2019.00078. Citado na p. 16.

MANI, P. K.; MANDAL, A.; BISWAS, S.; SARKAR, B.; MITRAN, T.; MEENA, R. S. Remote Sensing and Geographic Information System: A Tool for Precision Farming. In: **Geospatial Technologies for Crops and Soils**. Edição: Tarik Mitran, Ram Swaroop Meena e Abhishek Chakraborty. Singapore: Springer Singapore, 2021. P. 49–111. ISBN 978-981-15-6864-0. DOI: 10.1007/978-981-15-6864-0_2. Disponível em: <https://doi.org/10.1007/978-981-15-6864-0_2>. Citado na p. 17.

MATESE, A.; TOSCANO, P.; DI GENNARO, S. F.; GENESIO, L.; VACCARI, F. P.; PRIMICERIO, J.; BELLI, C.; ZALDEI, A.; BIANCONI, R.; GIOLI, B. Intercomparison of UAV, Aircraft and Satellite Remote Sensing Platforms for Precision Viticulture. **Remote Sensing**, v. 7, n. 3, p. 2971–2990, 2015. ISSN 2072-4292. DOI: 10.3390/rs70302971. Disponível em: <https://www.mdpi.com/2072-4292/7/3/2971>. Citado na p. 18.

MEGHAKUMAR. **CNN MODEL ARCHITECTURES WITH THEIR STRENGTHS AND WEAKNESSES**. 2021. Disponível em: <https://meghakumar245.medium.com/cnn-model-architectures-with-their-strengths-and-weaknesses-4cf81fc49cc2>. Citado na p. 27.

MELO, C. **Redes Neurais Multicamadas com Python e Keras**. 2019. [Acesso em: 10/04/2023]. Disponível em: <https://sigmoidal.ai/redes-neurais-python-keras-2/>. Citado na p. 21.

MOHIT DAYAL MEHAK GUPTA, M. G. e. a. Introduction to Machine Learning Methods With Application in Agriculture, 2023. DOI: https://doi.org/10.4018/978-1-6684-6413-7.ch012. Citado na p. 20.

ONYEMA, E.; BANJO, O.; UGAH, J.; AGUBOSIM, C. C.; OKE, O.; NWODO, O.; CALISTUS CHIDI, U. Prospects and Challenges of Precision Agriculture Technology in Rural Areas: A Case Study of Ubahu Community Enugu, Nigeria. **Journal of Computer Science and Its Application**, v. 28, p. 84–93, dez. 2021. DOI: 10.4314/jcsia.v28i2.10. Citado na p. 18.

PAN, S. J.; YANG, Q. A Survey on Transfer Learning. **IEEE Transactions on Knowledge and Data Engineering**, v. 22, n. 10, p. 1345–1359, 2010. DOI: 10.1109/TKDE.2009.191. Citado na p. 42.

REDMON, J. **Darknet: Open Source Neural Networks in C**. 2013–2016. http://pjreddie.com/darknet/. [Acesso em: 14/04/2023]. Citado na p. 32.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. **You Only Look Once: Unified, Real-Time Object Detection**. 2016. arXiv: 1506.02640 [cs.CV]. Citado nas pp. 32, 33.

REDMON, J.; FARHADI, A. **YOLOv3: An Incremental Improvement**. 2018. arXiv: 1804.02767 [cs.CV]. Citado na p. 33.

REN, S.; HE, K.; GIRSHICK, R.; SUN, J. **Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks**. 2016. arXiv: 1506.01497 [cs.CV]. Citado nas pp. 29, 30.

RIBERA, J.; GÜERA, D.; CHEN, Y.; DELP, E. J. **Locating Objects Without Bounding Boxes**. 2019. arXiv: 1806.07564 [cs.CV]. Citado na p. 49.

ROBOFLOW. **Roboflow**. https://www.roboflow.com/. [Acesso em: 14/04/2023]. Citado nas pp. 49, 50.

RUSSELL, S.; RUSSELL, S.; NORVIG, P.; DAVIS, E. **Artificial Intelligence: A Modern Approach**. Prentice Hall, 2010. (Prentice Hall series in artificial intelligence). ISBN 9780136042594. Disponível em: <https://books.google.com.br/books?id=8jZBksh-bUMC>. Citado na p. 20.

SCRIBBR. **Pearson Correlation Coefficient: Simple Definition, Calculation and Example**. 2022. [Acesso em: 30/05/2023]. Disponível em: <https://www.scribbr.com/statistics/pearson-correlation-coefficient/>. Citado na p. 78.

SEBE, N.; COHEN, I.; GARG, A.; HUANG, T. **Machine Learning in Computer Vision**. Springer Netherlands, 2005. (Computational Imaging and Vision). ISBN 9781402032745. Disponível em: <https://books.google.com.br/books?id=lemw2Rhr%5C_PEC>. Citado na p. 19.

SHAKHADRI, S. A. G. **Generate Your Own Dataset using GAN — analyticsvidhya.com**. https://www.analyticsvidhya.com/blog/2021/04/generate-your-own-dataset-using-gan/. [Acesso em: 26/05/2023]. Citado na p. 52.

SHORTEN, C.; KHOSHGOFTAAR, T. M. A survey on Image Data Augmentation for Deep Learning. **Journal of Big Data**, v. 6, n. 1, p. 60, jul. 2019. ISSN 2196-1115. DOI: 10.1186/s40537-019-0197-0. Disponível em: <https://doi.org/10.1186/s40537-019-0197-0>. Citado na p. 50.

SILVA, R. de; CIELNIAK, G.; GAO, J. **Towards agricultural autonomy: crop row detection under varying field conditions using deep learning**. arXiv, 2021. DOI: 10.48550/ARXIV.2109.08247. Disponível em: <https://arxiv.org/abs/2109.08247>. Citado na p. 46.

SO, S.; BADLOE, T.; NOH, J.; BRAVO-ABAD, J.; RHO, J. **Nanophotonics**, v. 9, n. 5, p. 1041–1057, 2020. DOI: doi : 10 . 1515 / nanoph - 2019 - 0474. Disponível em: <https : //doi.org/10.1515/nanoph-2019-0474>. Citado na p. 21.

SOARES, G. A.; ABDALA, D. D.; ESCARPINATI, M. C. Plantation Rows Identification by Means of Image Tiling and Hough Transform. In: VISIGRAPP. 2018. Citado na p. 44.

SZELISKI, R. **Computer Vision: Algorithms and Applications**. Springer International Publishing, 2022. (Texts in Computer Science). ISBN 9783030343729. Disponível em: <https://books.google.com.br/books?id=QptXEAAAQBAJ>. Citado na p. 19.

TAN, M.; LE, Q. V. **EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks**. 2020. arXiv: 1905.11946 [cs.LG]. Citado na p. 36.

TERVEN, J.; CORDOVA-ESPARZA, D. **A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond**. 2023. arXiv: 2304.00501 [cs.CV]. Citado na p. 40.

TILMAN, D.; BALZER, C.; HILL, J.; BEFORT, B. L. Global food demand and the sustainable intensification of agriculture. **Proceedings of the National Academy of Sciences**, v. 108, n. 50, p. 20260–20264, 2011. DOI: 10.1073/pnas.1116437108. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.1116437108. Disponível em: <https://www.pnas.org/doi/abs/10.1073/pnas.1116437108>. Citado na p. 17.

TSANG, S.-H. **Review: RetinaNet — Focal Loss (Object Detection)**. Towards Data Science, 2019. [Acesso em: 10/05/2023]. Disponível em: <https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4>. Citado na p. 31.

TSOUROS, D.; TRIANTAFYLLOU, A.; BIBI, S.; SARIGIANNIDIS, P. Data Acquisition and Analysis Methods in UAV- based Applications for Precision Agriculture. In: p. 377–384. DOI: 10.1109/DCOSS.2019.00080. Citado na p. 18.

ULTRALYTICS. **Ultralytics Documentation**. 2023. [Acesso em: 24/04/2023]. Disponível em: <https://docs.ultralytics.com/>. Citado na p. 37.

ULTRALYTICS, G. **Issue 189**. GitHub, 2023a. https://github.com/ultralytics/ultralytics/issues/189. Citado na p. 38.

ULTRALYTICS, G. **Ultralytics Repository**. GitHub, 2023b. https://github.com/ultralytics/ultralytics. Citado nas pp. 38, 57.

WANG, C.-Y.; BOCHKOVSKIY, A.; LIAO, H.-Y. M. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. **arXiv preprint arXiv:2207.02696**, 2022. Citado na p. 35.

WONG, K.-Y. **YOLOv7: YOLOv7 - Real-time Object Detection**. GitHub, 2021. https://github.com/WongKinYiu/yolov7. [Acesso em: 20/04/2023]. Citado na p. 35.

WU, Y.; KIRILLOV, A.; MASSA, F.; LO, W.-Y.; GIRSHICK, R. **Detectron2**. 2019a. https://github.com/facebookresearch/detectron2. Citado na p. 55.

WU, Y.; KIRILLOV, A.; MASSA, F.; LO, W.-Y.; GIRSHICK, R. **Detectron2: A PyTorch-based modular object detection library**. Facebook, 2019b. https://ai.facebook.com/blog/-detectron2-a-pytorch-based-modular-object-detection-library-/. Citado na p. 52.

XIE, S.; GIRSHICK, R.; DOLLÁR, P.; TU, Z.; HE, K. **Aggregated Residual Transformations for Deep Neural Networks**. 2017. arXiv: 1611.05431 [cs.CV]. Citado na p. 35.

ZHANG, Z.; HE, T.; ZHANG, H.; ZHANG, Z.; XIE, J.; LI, M. **Bag of Freebies for Training Object Detection Neural Networks**. 2019. arXiv: 1902.04103 [cs.CV]. Citado na p. 36.

Apêndices

Apêndice A – YOLOv8x Otimizada

A.1 *Batches*

As Figuras A.53, A.54, A.55 fornecem um exemplo dos *batches* utilizado durante o treinamento, validação e teste do modelo, respectivamente.

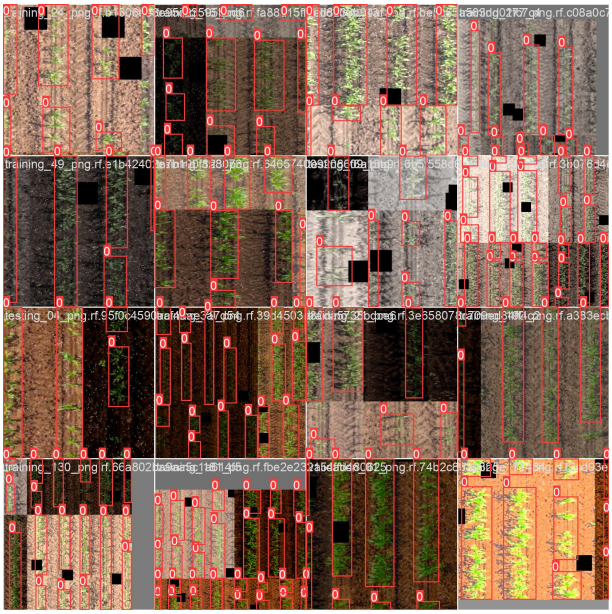


Figura A.53 – Exemplo de imagens utilizadas durante o treinamento

Fonte: autores

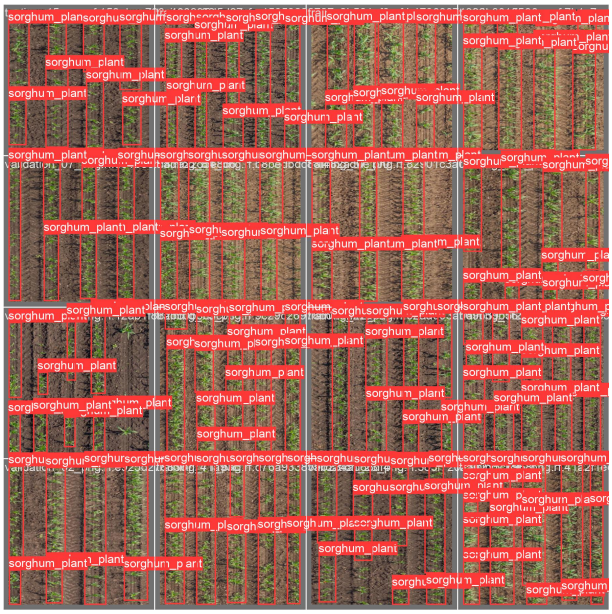


Figura A.54 – Exemplo de imagens utilizadas durante a validação

Fonte: autores

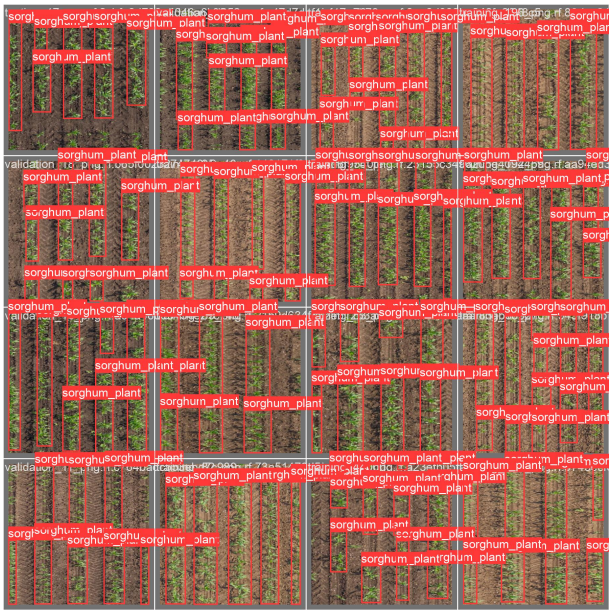


Figura A.55 – Exemplo de imagens utilizadas durante o teste

Fonte: autores

A.2 Resumo dos Resultados do Treinamento

A Figura A.56 contém diversos gráficos mostrando o desempenho da rede durante o treinamento.

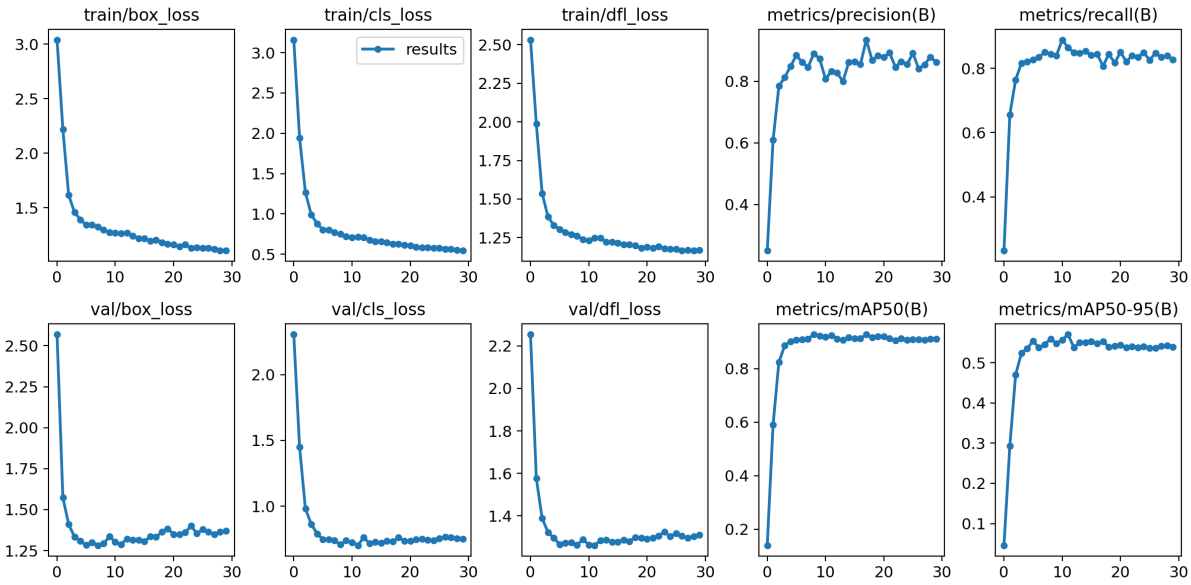


Figura A.56 – Gráficos do treinamento da YOLOv8x otimizada

Fonte: autores

A.3 Gráficos de Confiança e de *Precision x Recall* do Modelo

A Figura A.57 mostra os gráficos das métricas da rede em relação a confiança, de forma a verificar a confiança ideal para se obter o maior valor de determinada métrica. Já a Figura A.58 apresenta a curva de *Precision x Recall*, a qual fornece o valor de AP do modelo.

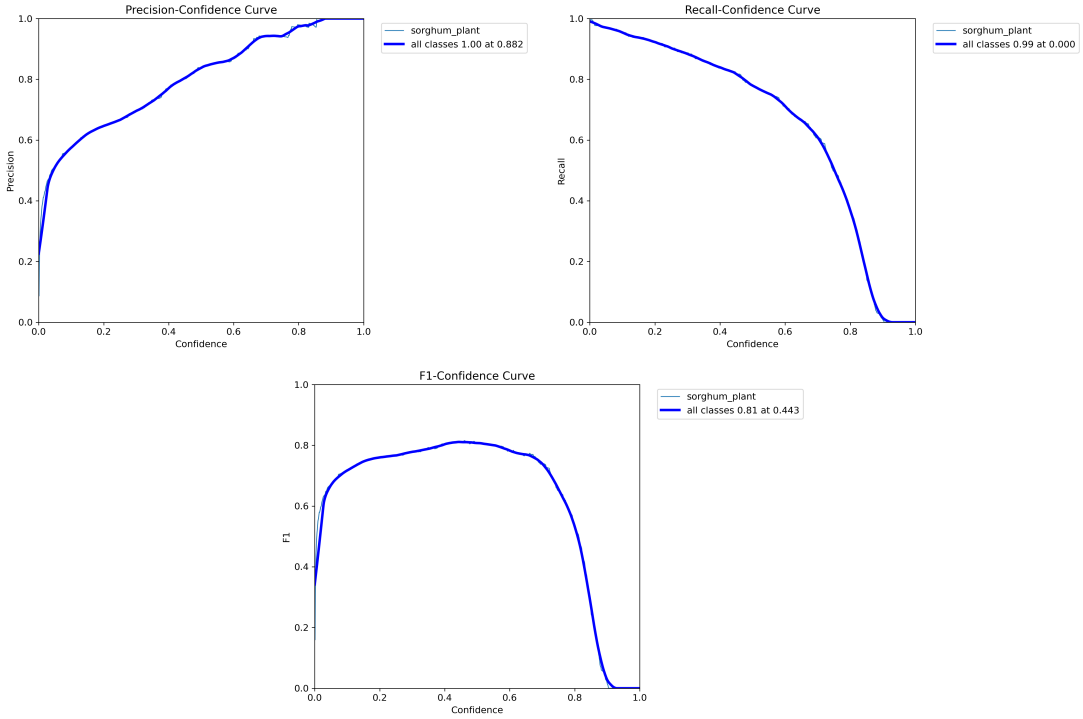


Figura A.57 – Gráficos de *Precision* x Confiança, *Recall* x Confiança e *F1-score* x Confiança

Fonte: autores

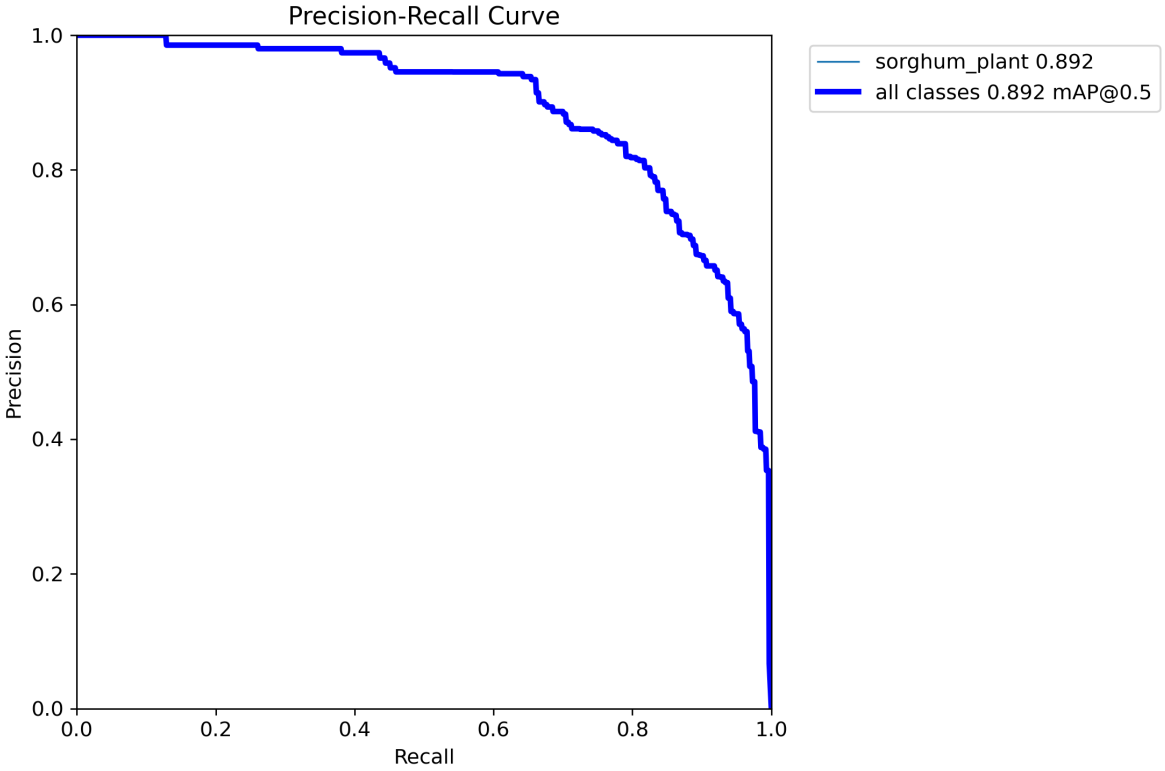


Figura A.58 – Gráfico de *Precision x Recall*

Fonte: autores

Apêndice B – Implementação dos Algoritmos em Python

B.1 Identificação das Falhas

Código B.1 – Código em Python para mostrar as falhas das linhas de plantação na imagem

```python
# Importando as bibliotecas necessarias
import os
import cv2
import numpy as np
from PIL import Image, ImageDraw, ImageFont
import re

# Funcao para desenhar linhas entre dois pontos e anotar o comprimento da linha
def draw_line(draw, start, end, fill=(0, 0, 255), width=3, font=None):
    # Desenhar a linha
    draw.line([start, end], fill=fill, width=width)
    # Calcular o comprimento da linha
    line_length = int(((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2) ** 0.5)
    # Determinar a posicao do texto
    text_position = (end[0] + 5, (start[1] + end[1]) / 2 - 10)
    # Anotar o comprimento da linha na imagem
    draw.text(text_position, f"{line_length}px", font=font, fill=fill)

# Funcao para agrupar caixas delimitadoras em colunas
def find_columns(boxes):
    columns = []
    for box in boxes:
        box_added = False
        for column in columns:
            if is_vertically_aligned(box, column[0]):
                column.append(box)
                box_added = True
                break
        if not box_added:
            columns.append([box])
    return columns

# Funcao para verificar se duas caixas estao alinhadas verticalmente
def is_vertically_aligned(box1, box2):
```

```
35      x1_1, _, x2_1, _ = box1
36      x1_2, _, x2_2, _ = box2
37      return x1_1 <= x2_2 and x1_2 <= x2_1
38
39  # Configuracao da fonte para anotar as imagens
40  font_path = "arial.ttf"
41  font_size = 18
42  font = ImageFont.truetype(font_path, font_size) if font_path else
        None
43
44  # Obter uma lista de todos os caminhos de imagem no diretorio
        especificado
45  folder_path = "imagens_areas_calculadas"
46  image_paths = [os.path.join(folder_path, file) for file in
        os.listdir(folder_path) if file.endswith('.jpg')]
47
48  # Funcao para extrair o numero da imagem
49  def extract_number(s):
50      return int(re.search(r'(\d+)_area', s).group(1))
51
52  # Ordenar os caminhos de imagem pela sequencia numerica extraida
        do nome do arquivo
53  image_paths = sorted(image_paths, key=extract_number)
54
55  # Definindo a pasta de saida para as imagens processadas
56  output_folder = 'imagens_falhas_total'
57
58  # Criar a pasta de saida se ela nao existir
59  if not os.path.exists(output_folder):
60      os.makedirs(output_folder)
61
62  # Lista para armazenar as distancias totais calculadas para cada
        imagem
63  total_distances = []
64
65  # Loop para processar cada imagem e caixas delimitadoras
        correspondentes
66  for i, (tensor, image_path) in enumerate(zip(bboxes_xyxy,
        image_paths)):
67      # Ler a imagem
68      image = cv2.imread(image_path)
69      pil_image = Image.fromarray(image)
70      draw = ImageDraw.Draw(pil_image)
71
72      # Converter tensor de caixas delimitadoras para numpy
73      boxes = tensor.numpy()
74      # Agrupar caixas em colunas
75      columns = find_columns(boxes)
76
77      total_distance = 0
78
```

```
79      # Para cada coluna de caixas, calcular e desenhar linhas entre
            caixas e entre caixas e bordas da imagem e adicionar todas
            as distancias a distancia total
80      for column in columns:
81          # Ordena as caixas na coluna por sua coordenada y1 (topo)
82          column = sorted(column, key=lambda box: box[1])
83
84          # Obtem a caixa superior (topo) e inferior (fundo) da
                coluna
85          top_box = column[0]
86          bottom_box = column[-1]
87
88          # Desenha uma linha do centro da caixa superior ate a
                borda superior da imagem
89          avg_x_top = (top_box[0] + top_box[2]) / 2  # media de x
                (coordenadas laterais) para a caixa superior
90          line_start_top = (avg_x_top, 0)  # inicio da linha (no
                topo da imagem)
91          line_end_top = (avg_x_top, top_box[1])  # fim da linha (na
                caixa superior)
92          draw_line(draw, line_start_top, line_end_top, fill=(0, 0,
                255), width=3, font=font)  # chama a funcao para
                desenhar a linha
93          total_distance += int(((line_end_top[0] -
                line_start_top[0]) ** 2 + (line_end_top[1] -
                line_start_top[1]) ** 2) ** 0.5)  # adiciona o
                comprimento da linha a distancia total
94
95          # Desenha uma linha do centro da caixa inferior ate a
                borda inferior da imagem
96          avg_x_bottom = (bottom_box[0] + bottom_box[2]) / 2  #
                media de x (coordenadas laterais) para a caixa inferior
97          line_start_bottom = (avg_x_bottom, bottom_box[3])  #
                inicio da linha (na caixa inferior)
98          line_end_bottom = (avg_x_bottom, image.shape[0])  # fim da
                linha (na borda inferior da imagem)
99          draw_line(draw, line_start_bottom, line_end_bottom,
                fill=(0, 0, 255), width=3, font=font)  # chama a funcao
                para desenhar a linha
100         total_distance += int(((line_end_bottom[0] -
                line_start_bottom[0]) ** 2 + (line_end_bottom[1] -
                line_start_bottom[1]) ** 2) ** 0.5)  # adiciona o
                comprimento da linha a distancia total
101
102         # Desenha linhas entre caixas adjacentes na mesma coluna
103         for j in range(len(column) - 1):
104             box1 = column[j]  # caixa superior
105             box2 = column[j + 1]  # caixa inferior
106             avg_x = (box1[0] + box1[2] + box2[0] + box2[2]) / 4  #
                    media de x (coordenadas laterais) para as duas
                    caixas
```

```
107             line_start = (avg_x, box1[3])  # inicio da linha (na
                    caixa superior)
108             line_end = (avg_x, box2[1])  # fim da linha (na caixa
                    inferior)
109             draw_line(draw, line_start, line_end, fill=(0, 0,
                    255), width=3, font=font)  # chama a funcao para
                    desenhar a linha
110             total_distance += int(((line_end[0] - line_start[0])
                    ** 2 + (line_end[1] - line_start[1]) ** 2) ** 0.5)
                    # adiciona o comprimento da linha a distancia total
111
112     # Adicionar distancia total a lista
113     total_distances.append(total_distance)
114
115     # Escrever a distancia total na imagem
116     draw.text((5, 25), f"Falha: {total_distance}px",
            font=ImageFont.truetype("arial.ttf", 18), fill=(255, 255,
            255))
117
118     # Converter a imagem do PIL de volta para numpy
119     image = np.array(pil_image)
120     # Definir o caminho de saida para a imagem processada
121     output_image_path = os.path.join(output_folder,
            f"image_{i}_falhas.jpg")
122     # Salvar a imagem processada
123     cv2.imwrite(output_image_path, image)
```

B.2 Cálculo da Área Cultivada

Código B.2 – Código em Python para calcular a área cultivada em cada imagem

```
1  from PIL import Image, ImageDraw, ImageFont
2  import os
3  import cv2
4  import numpy as np
5  from google.colab.patches import cv2_imshow
6  import csv
7
8  # Caminho do diretorio onde estao as imagens
9  folder_path = "inferencias"
10
11 # Tamanho total da imagem em pixels
12 image_size = 416 * 416
13
14 # Cria uma lista com todos os caminhos para as imagens no diretorio
15 image_paths = [os.path.join(folder_path, file) for file in
       os.listdir(folder_path) if file.endswith('.jpg')]
16
17 # Ordena a lista de caminhos de imagens
18 image_paths.sort()
```

```
19
20 # Define o diretorio de saida para as imagens processadas
21 output_folder = 'imagens_areas_calculadas'
22
23 # Cria uma lista vazia para armazenar os valores de cobertura de
       cada imagem
24 coverage_values = []
25
26 # Itera sobre a lista de tensores e caminhos de imagem
27 for i, (tensor, image_path) in enumerate(zip(bboxes_xywh,
       image_paths)):
28     total_area = 0
29     # Calcula a area total das caixas delimitadoras
30     for bbox in tensor:
31         x, y, w, h = bbox
32         area = w * h
33         total_area += area
34
35     # Calcula a porcentagem da imagem coberta pelas caixas
           delimitadoras
36     coverage = (total_area / image_size) * 100
37     coverage_values.append(coverage)
38
39     # Carrega a imagem usando OpenCV
40     image = cv2.imread(image_path)
41
42     # Converte a imagem OpenCV em uma Imagem PIL
43     pil_image = Image.fromarray(image)
44
45     # Sobrepoem a porcentagem na imagem usando Pillow
46     draw = ImageDraw.Draw(pil_image)
47     font_path = "arial.ttf"  # Caminho para a fonte
48     font_size = 18  # Ajuste o tamanho da fonte
49     font = ImageFont.truetype(font_path, font_size)
50     text = f"Area Cultivada: {coverage:.2f}%"
51     text_position = (5, 5)  # Ajuste a posicao do texto (x, y)
52     text_color = (255, 255, 255)  # Cor do texto em branco
53
54     draw.text(text_position, text, fill=text_color, font=font)
55
56     image = np.array(pil_image)
57
58     # Salva a imagem com o texto sobreposto
59     output_image_path = os.path.join(output_folder,
           f"image_{i}_area.jpg")
60     cv2.imwrite(output_image_path, image)
61
62     # Exibe a imagem
63     cv2_imshow(image)
64
65 # Define o caminho de saida para o arquivo csv que armazenara os
       valores de cobertura
```

```
output_csv_path = "coverage_values.csv"

# Escreve os valores de cobertura no arquivo csv
with open(output_csv_path, "w", newline="") as csvfile:
    csv_writer = csv.writer(csvfile)
    csv_writer.writerow(["Image", "Coverage"])
    for i, coverage in enumerate(coverage_values):
        csv_writer.writerow([f"image_{i}_area.jpg",
           float(coverage.item())])
```